VP Grundlagen Compilersysteme 29.6.2016

Team: RSQ – Moritz Quotschalla, Armin Rekic, Hana Salihodzic

✓ **Assignment 1:** We implemented the four MIPS instructions (sll, srl, sllv and srlv) for logical bitwise shifting according to the MIPS standard.

✓ **Assignment 2:** Extended the scanner and the parser by two new symbols which were necessary to implement the bitwise shift operators. We also modified the grammar by adding a new rule for shift expressions.

✓ **Assignment 3**: In order to simplify the left/rightShift procedures we replaced the former computation of shifting with the operators implemented in the last two assignments. "twoToThePowerOf" isn't called anymore in this functions → still self-compiling without issues.

✓ **Assignment 4**: For this assignment we introduced a local attribute (containing two memory cells – one for the value and one for the flag if constant or not) which was needed to perform constant folding on arithmetic expressions. Instead of loading integers within expressions which can be folded during compile time we delayed code generation and only loaded the folded value. In order to be able to fold expressions with more than two foldable factors in between two variables (e.g. x + 1 + 2 + 3 + 4 + y) we introduced a new register access function (secondPreviousTemporary()) because only then we could access the first variable. For this assignment we also had to extend the binary length. Constant folding works with all expressions and is self-compiling.

✓ **Assignment 5:** First we added a new rule in the C* grammar describing the data structure of an array. Then we implemented parsing and code generation and modified the symbol table by adding a new field for the size of the array. The array indices support constant folding. Whenever we have array as parameters of a procedure only a reference to the data structure is passed.

✓ **Assignment 6:** Analog to assignment 5. We added a new field in the symbol table for the size of the second dimension in order to be able to calculate the addresses of 2D-arrays properly. All kind of operations on 1D-/2D-array working including arrays as procedure parameter.

✓ **Assignment 7:** Modified the grammar by introducing a new datatype: STRUCT and adding two new rules, one for the struct declaration and one for the access.

✓ **Assignment 8:** Added two new fields in the symbol table to make struct access possible. In one field we stored the name of the struct to which the instance refers to and in the other we store a pointer to its struct table. So, for every new definition of a struct we first store the name in the symbol table and then we create a new struct table where we store all the fields of that record. Whenever we want to access a certain field from the record, we just iterate through the struct table and search for it. Note: We were not able to implement a method to get the size of a record (stored in symbol table entry) for allocating sufficient memory so we had to do this by hand (e.g. malloc(3 * WORDSIZE);).

✓ **Assignment 9:** Implemented boolean operators with lazy evaluation. Therefore, we created two fixup chains (FChain and TChain) where we store the jump addresses.

✓ **Assignment 10:** In order to make mixed expressions work we fixup addresses every time a chain of &&-Expressions ends since AND has stronger precedence than OR.

✓ **Assignment 11:** We constructed a free list in the emulated memory to keep track of the fixed size freed symbol table entries. So before allocating space for a symbol table entry we iterate the free list and check whether there is freed space at a previous used address.
Note: free doesn't work with symbol table entries. When implementing our solution in the selfie version before structs and boolean everything works as expected. We weren't able to figure out where our problem is.