Team AAL

Alexander Kollert, Lex Winandy, Andreas Eckschlager

Assignment 0: The Team **Working**
We inserted the `print()` right before the if for the compile process
Assignment 1: Shift Instructions **Working**
We implemented the machine instructions to use the build in functions of *selfie* to enable it to use bit wise shifting.
Assignment 2: Shift Operators **Working**
In this Assignment we added symbols to scanner and parser to enable the scanning and parsing of `>>` and `<<` for the shift operations
Assignment 3: Shift Operators Code Generation **Working**
We implemented the C operators for C* to work with shift operations, here we had to fiddle out, that the right shift is only a logic shift on positive numbers. Our wrap around for this was, do check whether the number is negative, if so, we check the amount to be shifted and if it is `< 31` we need to do the right shift without the sign bit and insert the sign bit again after the operation.
Assignment 4: Constant Folding **Working**
This assignment was a little bit tricky, we passed an argument throughout the parser routines, to indicate whether or not the past parsed expression is a constant. So that if we found a non constant part in the expression, it would be loaded into the registers, otherwise we continued to calculate the result at compile time. Also we implemented that, a expression like `a + 2 + 1` would be folded to `a + 3` although it was not asked.
Assignment 5: Arrays **Working**
First we added the symbols and edited the grammar, so it would work with arrays as well. Then we added two extra fields in the Symbol Table to store the size of the array as well as the array base-type (type of the content of the array). Then we made the correct address calculations for row major addressing of the arrays, also we implemented that they could be passed through arguments.
Assignment 6: 2-dimensional Arrays **part. Working**
We hardcoded it, so that only 2 dimensional arrays would work, although it was possible to make multidimensional arrays possible. We did in fact the same as we did with one dimensional arrays, so we first extended the parser, scanner and grammar to work with 2 dimensional arrays. Then we implemented the address calculation. If a two-dimensional array is used as function parameter it gives you the reference. This is of course unexpected and should be considered unfinished.
Assignment 7 + 8: Structs **Working**
For `structs` we also extended the grammar and implemented a field list for all fields of a `struct` definition. We also made it possible for self referencing `structs`, as we went back after we parsed the fields of the `struct` definition and passed it to the field with the self referencing `struct`. Our parser does not support access to a field of a field in <u>one</u> instruction e.g. `mystruct -> field1 -> field2.`
Assignment 9 + 10: Lazy Evaluation **Working**
We added a list for the jump addresses so that we could branch to the respective branch in the syntax tree. We also joined them for mixed boolean expressions. So that if we have a mixed expression, it jumps to the next boolean operator that is not the current operator.
Assignment 11: Memory Managment **Working**
We added a list of free memory blocks inside *mipster*, where we add the address of the freed element. We implemented it like a stack, so the last address that got inserted into the list is the first address that gets reused from malloc. This is possible because `free()` frees chunks of fixed size of `12 * WORDSIZE`.