

Assignment 1: Funktioniert

Funktionen für Shiftoperationen in MIPS eingefügt unter Verwendung von `DecodeRFormat`, um Informationen aus der Instruktion zu extrahieren. Implementierung von `getShamt()`, die den Shiftwert aus dem Immediate-Teil der Instruktion holt. Optionalen Debug output zu Funktionen hinzugefügt.

Assignment 2: Funktioniert

Scanner, Parser erweitern, dass die Shiftoperatoren `<<`, `>>` erkannt werden. Dazu die entsprechenden Symbole eingefügt. Grammar um Shiftoperationen erweitert (*gr_shiftExpression*). Präzedenz zwischen *gr_expression* und *gr_simpleExpression*. Jeweils Wert links und rechts vom Operator durch Aufruf von *gr_simpleExpression* ausgewertet und code für die Entsprechende Shift-Fuktion generiert.

Assignment 3: Funktioniert

Code-Generation bereits bei Assignment 2 implementiert. Shift-Operatoren in `leftShift()`- und `rightShift()`-Funktionen verwendet.

Assignment 4: Funktioniert

Parameter in `cstar()` erstellt und allen untergeordneten Grammar-Funktionen pointer auf diesen Parameter übergeben. Parameter speichert aktuellen Wert des Ausdrucks, falls dieser nur aus Konstanten besteht. Parameter enthält flag: Zeigt der jeweiligen Funktion, ob beide Operanden Konstanten sind und gefolgt werden können.

Assignment 5: Funktioniert teilweise

Eckigen Klammern zu Scanner hinzugefügt. Parser erweitert, damit Deklarationen und zugriffe auf Arrays erkannt werden (*gr_statement*: Schreiben, *gr_factor*: Lesen, *gr_cstar*: globale Deklaration, *gr_variable*: lokale Deklaration). Klasse `ARRAY` hinzugefügt. `SymbolTable` um Größe-Feld erweitert. Bei der Deklaration werden Einträge in der jeweiligen `symbolTable` mit dem entsprechenden Scope, Offset und der jeweiligen Größe erstellt. `SymbolTable`-Suche erweitert, damit auch Arrays gefunden werden.

Assignment 6: nicht implementiert

Assignment 7: Funktioniert

Neue Funktionen *gr_struct_dec* für Deklaration und *gr_struct_def* für Definition implementiert. Fallunterscheidung ob Deklaration oder Definition in *gr_cstar* und *gr_procedure*. Klassen `STRUCT_DEF` für Struct-Definitionen und `STRUCT_F` für Struct-Felder eingeführt. Struct Definition \rightarrow `STRUCT_DEF`-`SymbolTable` Eintrag \rightarrow für jedes Feld `STRUCT_F`-`SymbolTable`-Eintrag. `SymbolTable` um `belongsTo` Eintrag erweitert, um Felder Definitionen zuordnen zu können. `SymbolTable`-Suche erweitert, damit auch Structs gefunden werden. Deklarierter Struct hat Klasse "VARIABLE".

Assignment 8: Funktioniert teilweise

ÄrrowOperator zu Scanner hinzugefügt. Neue Funktion *gr_struct_acc* implementiert. Aufgerufen wenn in *gr_statement* (schreiben) oder *gr_factor* (lesen) ein Identifier gefolgt von Ärrowerkannt wird. Offset für Feld aus Scope, Offset der Struct-Variable und in `STRUCT_F`-`SymbolTableEntry` des jeweiligen Felds gespeichertem Offset errechnet. Globale und Lokale Definition sowie Deklaration und Zugriff funktionieren sowohl in Selfie, wie auch in diversen Test-Files. Implementierung der `SymbolTable`-Einträge als Struct funktioniert nicht.

Assignment 9: Funktioniert

Scanner erweiter, sodass UND, OR und NOT erkannt werden. Neue Grammar-Funktion *gr_boolExpression* erstellt (UND und OR Ausdrücke haben niedrigste Präzedenz, NOT die höchste). Not in *gr_factor* hinzugefügt. Wird NOT erkannt, wird Flag gesetzt. Bei Rückkehr zu *gr_boolExpression* wird für Ausdrücke der Form (Variable OP-Symbol Variable) das Operator-Symbol invertiert, wenn die Flag gesetzt ist. Variablen und Konstanten werden bei einem Wert > 0 als true, bei 0 als false gewertet und können auch invertiert werden. Branches nach jedem bool'schen Operator eingefügt \rightarrow springt zum Ende des statements, der if- oder while- Bedingung, sobald die Bedingung für Lazy Evaluation erfüllt ist. T- und F- Jump Listen speichern Adressen der Instructionen, für die ein Fixup durchgeführt werden soll.

Assignment 10: Funktioniert

Gemischte UND und OR Ausdrücke. *gr_boolExpression* ersetzt durch: *gr_orExpression* (niedrigste Präzedenz) und *gr_andExpression* (zweit-niedrigste Präzedenz). Nach jedem Bool'schen Operator Branch generiert. Lazy Evaluation: Wenn aktuelles Ergebnis 1 und folgender Operator OR, dann Branch zum ende des Statements oder der if- oder while-Bedingung. Wenn aktuelles Ergebnis 0 und folgender Operator AND \rightarrow Sprung zum nächsten OR.

Assignment 11: Funktioniert

Neue Library Funktion "free"erstellt. Code für "free" wird, wie auch für alle anderen Library-Funktionen am Anfang ins binary geschrieben (durchgeführt durch `emitFree()`). Aufruf von `free()` im Code bewirkt einen Sprung zur Adresse, an der in `emitFree()` definierten Instruktionen stehen. Dadurch wird im Endeffekt ein `SYSCALL` mit der entsprechenden Nummer ausgelöst. `ImplementFree()` definiert Verhalten wenn ein `SYSCALL` mit Nummer 4042 gefunden wird. Adressen freier Speicherzellen in Free-List verwaltet. Jeder Eintrag zeigt auf den nächsten. Gespeichert an der ersten Speicherzelle des jeweiligen gefreeten Blocks im Heap. Malloc angepasst, dass für passende Speichergrößen zuerst der erste Free-List Eintrag verwendet. Passt die Größe nicht \rightarrow normale Bump-Pointer-Allocation.

Assignment 12: Funktioniert

Code-Cleanup.

siehe Assignment_12.pdf