

Technische Universität Braunschweig
Programmiersprachen und Übersetzer

Prof. Dr. Christan Dietrich

_____	_____	_ _ _ _ _ _ _ _ _
(Name)	(Vorname)	(Matrikel-Nr.)
_ _ _ _ _ _ _ _ _	_____	_____
(HBK Matrikel-Nr.)	(Studiengang)	(Semester)

Durch meine Unterschrift bestätige ich

- den Empfang der vollständigen Klausur (21 Seiten inklusive Deckblatt),
- die Kenntnisnahme der Hinweise auf Seite 2.

Braunschweig, 19.08.2024

(Unterschrift)

Hinweise

Bitte lesen Sie die folgenden Informationen aufmerksam!

- Die Bearbeitungszeit beträgt 90 Minuten. Antworten können in deutscher oder englischer Sprache verfasst werden.
- Als Hilfsmittel sind **ausschließlich** Lineal, Stift und ein nicht-programmierbarer Taschenrechner zugelassen.
- Die Lösung einer Aufgabe soll auf das Aufgabenblatt in den dafür vorgesehenen Raum geschrieben werden. Beachten Sie bitte, dass der freigelassene Platz großzügig bemessen ist und nicht unbedingt der erwarteten Antwortlänge entspricht. Sollte der Platz nicht ausreichen, können Sie die Rückseiten der Aufgabenblätter mitverwenden. Kennzeichnen Sie dabei die Zugehörigkeit Ihrer Lösung zu einer Aufgabe.
- Bei Bedarf können zusätzliche, besonders markierte, Lösungsblätter (weiß) ausgeteilt werden. Vermerken Sie vor deren Verwendung unbedingt den Klausurcode darauf!
- Die Lösungen müssen dokumentenecht in blau oder schwarz geschrieben werden. Als falsch Erkanntes muss deutlich durchgestrichen werden. Tintenkiller und andere Korrekturstifte dürfen nicht verwendet werden. Keinen Bleistift verwenden!
- Schmierpapier (farbige Blätter) darf **nicht** abgegeben werden. Bei Bedarf ist von der Aufsicht weiteres Schmierpapier (farbig) erhältlich.
- Es dürfen **keine nummerierten** Seiten herausgetrennt werden. Die letzte Seite enthält den Klausurcode und darf herausgetrennt werden
- Aktion:** Tragen Sie Ihren Namen und Vornamen, Ihre Matrikelnummer, Studiengang und Fachsemesterzahl auf dem Deckblatt der Klausur ein.
- Aktion:** Überprüfen Sie die Anzahl der Blätter Ihrer Klausur und unterschreiben Sie die Erklärung auf dem Deckblatt.
- Aktion:** Bitte legen Sie Ihren gültigen Studierenden- und Lichtbildausweis zur Kontrolle bereit.
- Um Unruhe und die Störung Ihrer Mitstudierenden zu vermeiden, ist die vorzeitige Abgabe der Klausur ausgeschlossen. Bleiben Sie an Ihrem Platz sitzen, bis die Aufsicht das Zeichen zum Gehen gibt.

Viel Erfolg!

Korrektur – Zusammenfassung

	erreichbare Punkte	erhaltene Punkte					
		1	2	3	4	5	
Aufgabe 1	15						
Aufgabe 2	17	1	2				
Aufgabe 3	15	1	2				
Aufgabe 4	8	1					
Aufgabe 5	12	1					
Aufgabe 6	16	1	2				
Aufgabe 7	7	1					
Summe	90						

Aufgabe 1: Ankreuzfragen (15 Punkte)

In dieser Aufgabe sind jeweils m Aussagen angegeben. Davon sind n ($0 \leq n \leq m$) Aussagen richtig. Kreuzen Sie jeweils an, ob die entsprechende Aussage richtig oder falsch ist.

Jede korrekte Antwort gibt 0,5 Punkte, jede falsche Antwort 0,5 Punkte Abzug. Nicht beantwortete Aussagen gehen neutral in die Bewertung ein. Eine Teilaufgabe wird minimal mit 0 Punkten gewertet, falsche Antworten wirken sich nicht auf andere Teilaufgaben aus.

Wollen Sie eine falsch angekreuzte Antwort korrigieren, streichen Sie bitte das Kreuz mit drei waagrechten Strichen durch (~~☒~~).

Lesen Sie die Frage genau, bevor Sie antworten.

Aufgabe 1.1: Syntaktische Analyse

3 Punkte-

Richtig Falsch

- Entscheidungen die der Parser nicht treffen kann, verschiebt er in die Lexikalische Analyse.
- Die Initialisierungsphase eines Übersetzerlaufs heißt Bootstrapping.
- LL(1)-Sprachen können deterministisch geparkt werden.
- Der Kontrollflussgraph deckt alle Pfade durch eine Funktion ab.
- Nach der syntaktischen Analyse wissen wir, ob ein gültiges Programm vorliegt.
- Die Blockstruktur imperativer Sprachen ist ein rekursives Konstrukt.

Aufgabe 1.2: Semantische Analyse

3 Punkte-

Richtig Falsch

- Eine Closure ist eine Funktion mit (partiell) gebundenem Ausführungskontext.
- Die FIRST-Menge enthält die kompatiblen Funktionen bei einem überladenen Funktionsaufruf.
- Vom Wurzelnamensraum sind alle deklarierten Namen sichtbar.
- Eine Cast-Operation in C ist eine Form des Inklusionspolymorphismus.
- Record- und Arraytypen sind skalare Typen.
- Parameternamen sind innerhalb einer Funktion sichtbar.

Aufgabe 1.3: Codeerzeugung

Richtig Falsch

3 Punkte-

- Trotz existierender read-only Referenz auf ein Objekt, können Andere das Objekt verändern.
- Die Kanten im Kontrollflussgraphen entsprechen möglichen Übergängen zwischen einzelnen Basisblöcken.
- Mark-and-Sweep entfernt nur Objekte, auf die keine einzige Referenz zeigt.
- Die Initialisierung eines Objekts geht der Allokation des Objekts voraus.
- Ein Selektions-Statement provoziert eine Verzweigung im Kontrollfluss.
- Der L-Wert einer Variable ist ein Zeiger.

Aufgabe 1.4: Optimierung

Richtig Falsch

3 Punkte-

- Nach der Optimierung ist der C Ausdruck $4*13+4$ ist ebenso effizient wie 56.
- Das Alias-Problem tritt auf, wenn Typen ähnlich benannt sind.
- Der Optimierer lässt die Anzahl der Basisblöcke gleich.
- Eine gute Registerzuweisung reduziert Speicherzugriffe des Programms erheblich.
- Ein optimierender Übersetzer passt den Quellcode an.
- Konstantenfaltung ändert das beobachtbare Verhalten.

Aufgabe 1.5: Maschinencode

Richtig Falsch

3 Punkte-

- Beim Laden entsteht das BSS Segment durch eine Diskrepanz zwischen `FileSize` und `MemSize` beim Datensegment.
- Der Programmlader erzeugt ELF-Dateien.
- Bei der Relokation wird der ausführbare Binärcode verändert.
- Aufrufkonventionen sind spezifisch für ein Betriebssystem/Übersetzer.
- Ein ELF-Symbol gibt einem (oder mehreren) Bytes in einer Section einen Namen.
- Die Link-View einer ELF-Datei wird zur Laufzeit benötigt.

Aufgabe 2.2: PREDICT-Mengen

12 Punkte

Eine gegebene Grammatik hat die Terminale **Ret, If, Else, End, Now, \neg** und die Nichtterminale *stmt, ifStmt, mod, S*. Das Startsymbol ist *S* und die untenstehende Tabelle enthält die Produktionsregeln der Grammatik.

Bestimmen Sie die PREDICT-Mengen für die gegebene Grammatik!

Hinweis: Anhand von EPS können Sie entscheiden, ob Sie nur die FIRST- oder auch die FOLLOW-Menge berechnen müssen.

$A \rightarrow \alpha$	EPS(α)	FIRST(α)	FOLLOW(<i>A</i>)	PREDICT
$S \rightarrow stmt \neg$				
$stmt \rightarrow ifStmt$				
$stmt \rightarrow mod \mathbf{Ret}$				
$ifStmt \rightarrow \mathbf{If} stmt \mathbf{Else}$ $stmt \mathbf{End}$				
$ifStmt \rightarrow \epsilon$				
$mod \rightarrow \epsilon$				
$mod \rightarrow \mathbf{Now}$				

Aufgabe 3: Semantische Analyse (15 Punkte)

Aufgabe 3.1: Namensauflösung (Let-Sprache)

10 Punkte

Gegeben sei folgendes Programm in der let-Sprache, die Sie in der Vorlesung kennengelernt haben. Namen werden mittels **let** deklariert und mittels **ref** referenziert. Standardmäßig sind alle Namen intern sichtbar, extern sichtbare Namen sind gesondert mit **[E]** gekennzeichnet. Importierte Namen werden **nicht** re-exportiert.

Lösen Sie alle Namensreferenzen auf, indem Sie den **vollständig-qualifizierten Namen** der Deklaration oder **NOT FOUND** angeben (linker Kasten). Zusätzlich sollen Sie im jeweils rechten Kasten angeben, ob die Namensauflösung erfolgreich (OK) oder fehlerhaft (Fehler) war, falls bspw. ein Namen mit interner Sichtbarkeit referenziert wurde.

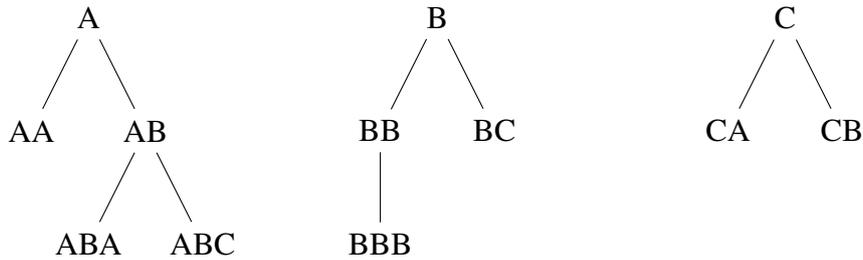
```

1 let A
2 ref A ----- ::A (Zeile 1)      OK
3 let[E] lib {
4   let[E] A
5   let[E] lib = {
6     let lib
7     let[E] PI
8     let[E] A
9     ref lib::ref A ----- ::
12 ref lib::ref PI ----- ::
14 }
15 let[E] math {
16   import lib::::*
17   let[E] PI
18   let baz
19   ref lib ----- ::
20   ref A ----- ::
21 }
22 let app = {
23   import ::lib::*
24   import ::math::*
25   ref A ----- ::
26   ref math:: ----- ::
27   ref PI ----- ::
28 }
29 ref lib:: ----- ::
  
```

Aufgabe 3.2: Aufruf einer überladenen Funktion

5 Punkte

Gegeben sei folgende Vererbungshierarchie, bei der A, B und C unabhängige Basisklassen sind.



Ihre Aufgabe ist es den überladenen Funktionsaufruf in Zeile 4 aufzulösen. Dabei soll ein **dynamischer Dispatch** im ersten und im letzten Argument durchgeführt werden.

```

1  A  arg1  = new ABA();
2  BB arg2  = new BBB();
3  C  arg3  = new CA();
4  BB retVal = foo(arg1, arg2, arg3);
  
```

Berechnen Sie zunächst die Argument-Signatur, welche Sie im nächsten Schritt für die Auflösung der Überladung verwenden. Die **grau hinterlegten Felder** bieten Platz für Ihre Antworten:

Um die Funktion mit der höchsten Spezifität zu finden, berechnen Sie für jede der folgenden Deklarationen den Malus. Sollte dies für eine Signatur nicht möglich sein, begründen Sie dies kurz.

Nr.	Parameter-Signatur	Malus	Problem bei der Malusberechnung
1.	foo(AB, BB, C)		
2.	foo(ABC, B, C)		
3.	foo(A, BB, CA)		
4.	fof(ABA, BB, C)		
5.	foo(ABA, B, C)		
6.	foo(AB, BB, CA)		

Geben Sie die Nummer (Nr.) der Funktion mit der höchsten Spezifität an oder -1, falls die Überladung nicht eindeutig auflösbar ist.

Funktion mit höchster Spezifität:

Platz für weitere Notizen.

Aufgabe 4: Garbage Collection (8 Punkte)

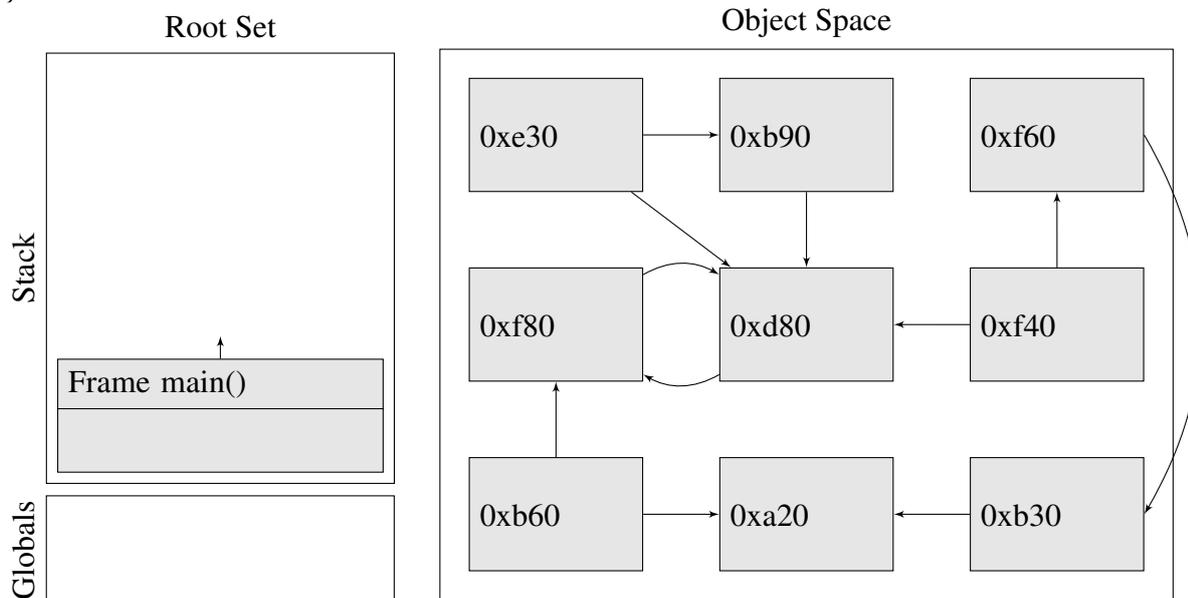
8 Punkte-

Gegeben sei folgendes Programm (in C-Syntax), welches zusammen mit einem Mark-and-Sweep Garbage Collector ausgeführt wird. Das Programm befindet sich aktuell in Ausführung und die unten stehende Grafik zeigt den Object Space, in den das Programm Referenzen durch C-Casts erzeugt. Nehmen Sie an, dass das Programm zum durch den Kommentar gekennzeichneten Zeitpunkt durch den Garbage Collector unterbrochen wird.

Bestimmen Sie zunächst das Root Set, inklusive der Verbindung in den Object Space, und führen Sie dann die Mark-Phase der Garbage Collection durch. Markieren Sie in der Grafik zu löschende Objekte mit einem Kreuz(×) und Objekte, die nicht gelöscht werden dürfen, mit einem Haken (✓). Die Prozessorregister können vernachlässigt werden.

```

1 void *R = (void *) 0xa20;
2
3 void *create_complex_object() {
4     return (void *) 0xe30;
5 }
6
7 void start(void *g, int v) {
8     void *l = (void *) 0xf40;
9     l = create_complex_object();
10    void *k = (void *) 0xd80;
11
12    // <-- Startzeitpunkt des Garbage Collectors!
13
14    void *a = (void *) 0xb90;
15 }
16
17 void main() {
18     void *g = (void *) 0xf80;
19     int l = 0xb30;
20     start(0xf80, 0xf80);
21 }
    
```



Aufgabe 5: Zwischencodeerzeugung (12 Punkte)

12 Punkte

Übersetzen Sie das gegebene Programm in der Sprache L0 manuell, **ohne** Umweg über einen abstrakten Syntaxbaum, zu IR-Code. Verwenden Sie dazu die 3-Addresscode -Notation, die Sie in der Vorlesung kennengelernt haben. Geben Sie für die Funktion foo ebenfalls die Menge der Parameter und die Menge der lokalen Variablen an.

Hinweis: Es lohnt sich, wenn Sie sich zuerst den Kontrollflussgraphen skizzieren!

```
func foo(limit : int, ptr : &int) : int {
  var ret : int; var i : int;
  ret := 0;      i := 0;
  while (i <= limit) {
    i := i + 1;
    if ((i / 2) * 2 == i) {
      continue;
    }
    ret := ret + i * (*ptr) + 3;
  }
  return ret;
}
```

Schreiben Sie Ihre Lösung auf die **nächste Seite** →

Platz für weitere Notizen.

Aufgabe 6: Optimierung (16 Punkte)

Im Folgenden sollen Sie zwei unterschiedliche Optimierungen durchführen, die Berechnungen und Zuweisungen entfernen, deren Ergebnis nicht benötigt wird. Bitte beachten Sie, dass Sie bisher nur die erste Variante exakt so aus der Vorlesung kennen!

Aufgabe 6.1: Fluss-INsensitive Dead Variable Elimination

Zunächst sollen Sie eine flussinsensitive Dead Variable Elimination auf dem folgenden Programm durchführen. Dabei dürfen Sie nur Instruktionen entfernen, deren Ergebnis niemals direkt oder indirekt von einer anderen Instruktion gelesen wird. Der Operand von Return gilt immer als gelesen.

7 Punkte

Führen Sie die Optimierung als Fixpunktiteration aus und markieren Sie in der jeweiligen Iteration welcher Befehl entfernt wurde. Jede Iteration verläuft in zwei Phasen: (1) Erkennung der ungenutzten Variablen und (2) Entfernung der unnötigen/toten Zuweisungen. Im folgenden Beispiel wurde der Befehl in der 3. Iteration als eine unnötige Zuweisung erkannt und entfernt:

IR-Befehl	Iteration							
	1	2	3	4	5	6	7	8
a := Add b, c			×	-	-	-	-	-

Hinweis: Bei flussinsensitiven Optimierung spielt der Kontrollfluss keine Rolle. Daher sind die Befehle des folgenden Programms von uns in zufälliger Reihenfolge notiert worden!

IR-Befehl	Iteration							
	1	2	3	4	5	6	7	8
d := Mul b, 3								
a := Assign 0								
b := Add b, 1								
ptr := Ref i								
k := Call fn, ptr								
c := Add a, 2								
h := Add e, g								
g := Sub d, b								
e := Div c, d								
j := Sub i, 4								
f := LessEqual a, 3								
i := Add i, i								
Return d								

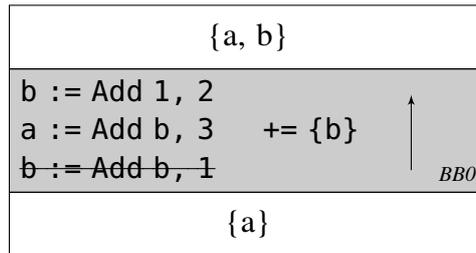
Platz für weitere Notizen.

Aufgabe 6.2: Fluss-Sensitive Dead Store Elimination

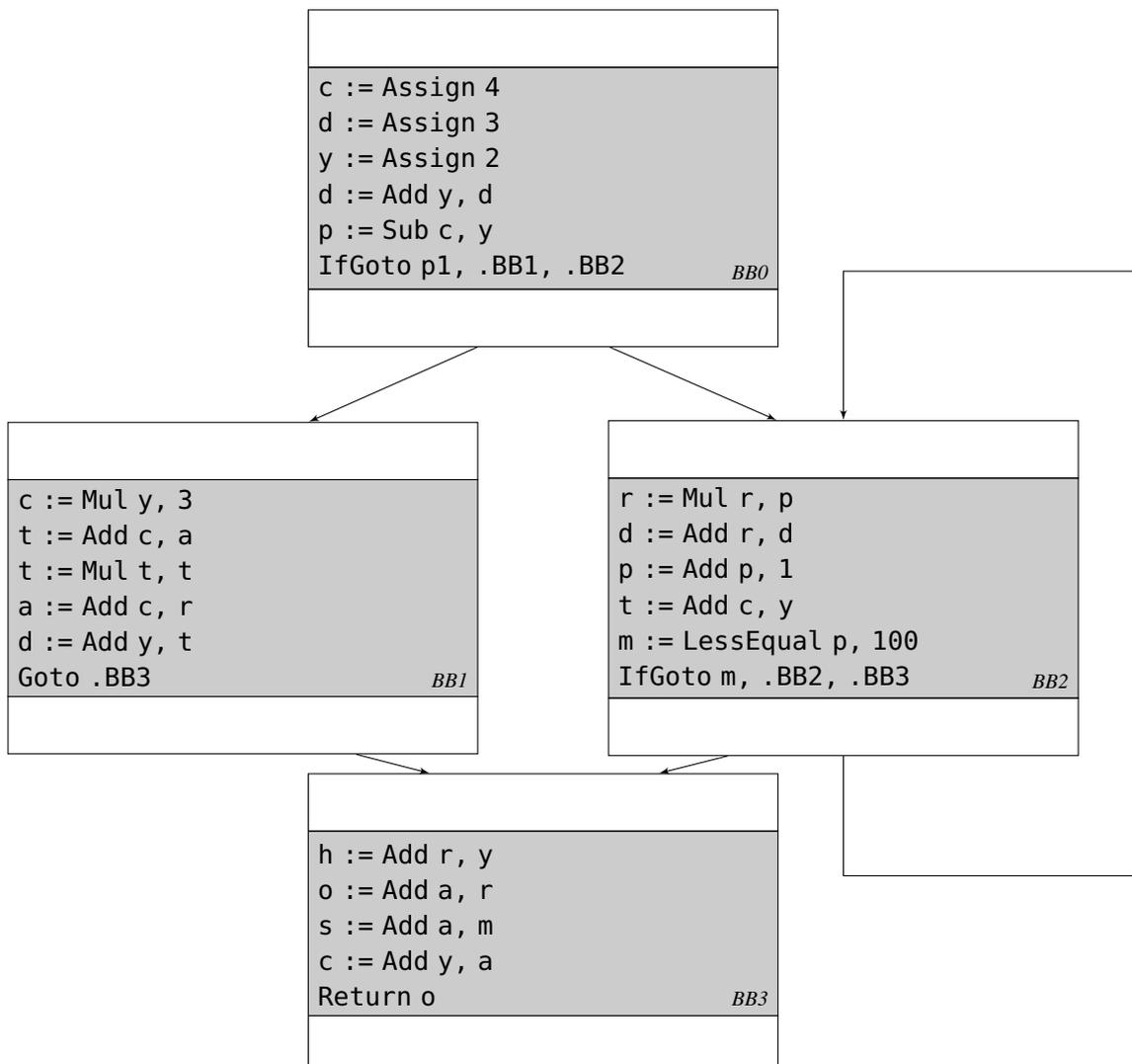
9 Punkte-

Im Folgenden, sollen Sie eine flusssensitive Dead Store Elimination als rückwärtsgerichtete Datenflussanalyse im Fixpunktverfahren durchführen. Dabei propagieren Sie die Menge der verwendeten Variablen “von hinten nach vorne” (im Kontrollflussgraph) und von “unten nach oben” (im Basisblock).

Eine Variable gilt als benutzt, wenn sie in einem Nachfolgerblock benutzt wird oder wenn sie als Operand für eine Instruktion dient, deren Ergebnis bereits als benutzt markiert ist. Der Rückgabewert wird immer als benutzt markiert. Das folgende Beispiel zeigt, wie die Dead Store Elimination die Menge der benutzten Variablen von hinten nach vorne propagiert und dabei Instruktionen entfernt, deren Ergebnis (noch) nicht als benutzt markiert wurde



Hinweis: Beginnen Sie mit ihrer Optimierung bei BB3 und arbeiten Sie von unten nach oben.



Platz für weitere Notizen.

Aufgabe 7: Registervergabe (7 Punkte)

Im Folgenden sollen Sie eine Registervergabe „mit Gedächtnis“ durchführen. Vervollständigen Sie dazu die Assemblerausgabe, indem Sie die Operanden der Makroexpansion und eventuell nötige Spill- und Ladebefehle ergänzen! Geben Sie weiterhin den internen Zustand des Registerallokators nach der letzten Assemblerinstruktion eines IR-Befehls aus! Schreiben Sie Ihre Lösung in die grau hinterlegten Felder. Die übrigen Felder können Sie für Notizen benutzen.

Randbedingungen der Registervergabe:

- **Aufrufkonvention:** Der erste Parameter eines Funktionsaufrufs muss nach %eax; der Rückgabewert ist nach der Rückkehr in %eax; der Aufgerufene sichert keinen Maschinenzustand.
- Nehmen Sie an, dass zu jeder Variable eine **Referenz** erzeugt wurde!
- Verwenden Sie **symbolische Variablennamen** anstatt %ebp-relativer Slotoffsets.

Dokumentation zu einigen Assemblerbefehlen:

mov %eax, %ebx	Kopiert den Inhalt des Registers %eax ins Register %ebx.
mov x, %eax	Kopiert den Inhalt des Variablenslots x ins Register %eax
add %eax, %ebx	Addiert beide Register; das Ergebnis landet in %ebx.
mov (%eax), %ebx	Dereferenziert den Zeiger in %eax; das Ergebnis landet %ebx.
mov %eax, (%ebx)	Kopiert den Inhalt von %eax an die Adresse in %ebx.
xchg %eax, %ecx	Tauscht die Inhalte der Register %eax und %ecx.

Aufgabe 7.1: Basisblock BB0

7 Punkte

IR-Code	Assemblerausgabe	Zustand nach Instruktion		
		%eax	%ebx	%ecx
b := Assign a	mov a, %eax	a, clean	---	---
	mov <input type="text" value="%eax"/> , <input type="text" value="%ebx"/>	a, clean	b, dirty	---
a := Add c, b	<input type="text"/>			
	<input type="text"/>			
	<input type="text"/>			
	add <input type="text"/> , <input type="text"/>			
c := Load *b	<input type="text"/>			
	<input type="text"/>			
	<input type="text"/>			
	mov (<input type="text"/>) , <input type="text"/>			
a := Call fn, d	<input type="text"/>			
	<input type="text"/>			
	<input type="text"/>			
	call fn			
b := Assign a	<input type="text"/>			
	<input type="text"/>			
	<input type="text"/>			
	mov <input type="text"/> , <input type="text"/>			

Platz für weitere Notizen.

Diese Seite darf herausgetrennt werden.

Diese Seite darf herausgetrennt werden.

Entfernen Sie dieses Blatt vor der Bearbeitung der Klausur und bewahren Sie es auf! Mit Hilfe des Codes können Sie Ihr persönliches Ergebnis online abfragen.

Klausur zur Vorlesung **Programmiersprachen und Übersetzer**, 2024-08-19.



Ergebnisse online: <https://www.ibr.cs.tu-bs.de/courses/ss24/psu>