Institut für Informatik
und Praktische Mathematik
der Universität Kiel

Kiel, 12.06.2002

Übungen zu Organisation und Architektur von Rechnern
SS 2002
Serie 7

Aufgabe 20 (10 Punkte)

Schreiben Sie in Assembler einen Interpreter für die in der Vorlesung vorgestellen SECDH-2 Instruktionen. Halten Sie sich dabei an folgende Vorgaben:

Darstellung der Stacks:

- Die Stacks S, E und D sind in separaten Speicherregionen darzustellen. Genauso wie der System-Stack wachsen alle Stacks in Richtung niedriger Adressen. Die dafür erforderlichen Speicherbereiche sind per `DS.L` Anweisung zu reservieren.

  Hinweis: Eine Überlauf-Kontrolle dieser Stacks ist nicht unbedingt erforderlich. Es sollte jedoch darauf geachtet werden, daß diese mit einer geeigneten Größe angelegt werden.

- Der Programmcode soll „hart verdrahtet“ sein, d. h. mit Hilfe der `DC.L` Anweisung sowie Mnemonischer Op-Codes (`EQU` Anweisung) beim Assembler-Code für den Interpreter stehen. Die Fakultätsfunktion sollte sich z. B. durch folgenden Assembler-Code darstellen lassen:

```
P_FAC  DC.L  ARGS,1,MKFRAME,1,PUSH_ES,0,0,PUSH_S,0,EQ,BRC,P_T,P_F,FREE,1,RET
P_T    DC.L  PUSH_S,1,RET
P_F    DC.L  PUSH_S,1,PUSH_ES,0,0,MINUS,LINK,0,CALL,P_FAC,PUSH_ES,0,0,MULT,RET
```

Codierung der SECDH-Programme:

- Jede Operation und jeder Operand einer Instruktion haben eine Länge von 32 Bit.
- Die verschiedenen Operationen sind gemäß folgender Tabelle zu codieren (Op-Codes):

PUSH_S	$00000000
PUSH_ES	$00000004
MKFRAME	$00000008
FREE	$0000000c
BRC	$00000010
CALL	$00000014
RET	$00000018
AP	$0000001c
PLUS	$00000020
MINUS	$00000024
MULT	$00000028
EQ	$0000002c
GT	$00000030
ARGS	$00000034
LINK	$00000038
CREATE_CLOS	$0000003c
DONE	$00000040

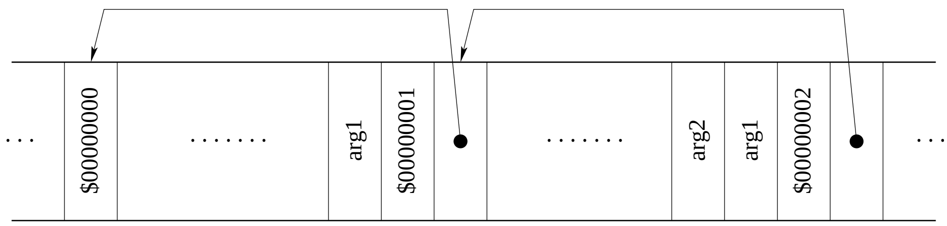
  Dabei stellt `DONE` eine neu einzuführende Operation dar, die das Ende der Berechnung kennzeichnet. Alle ggf. erforderlichen Parameter sind je nach Funktion entweder als 32-Bit Hexadezimal-Adressen oder als vorzeichenbehaftete Long-Werte zu interpretieren.

Codierung der S-Stack-Einträge:

- Zeiger auf Code-Fragmente, Closure-Strukturen oder Environment-Einträge liegen direkt als 32-Bit Adressen auf dem S-Stack.
- Vorzeichenbehaftete 32-Bit Werte liegen ebenfalls direkt auf dem S-Stack.
- Die Bool'schen Werte werden als `$00000003` für true und `$00000007` für false codiert.

Aufbau der Environments:

- Der E-Stack besteht ausschließlich aus Environment-Frames.
- Ein Environment-Frame besteht entweder aus einem NULL-Zeiger, d. h. `$00000000`, oder aus einem Zeiger auf das nächste Frame, gefolgt von einem vorzeichenbehafteten 32-Bit Wert, der die Anzahl der Elemente im Frame angibt:



Die Frame-Einträge (arg1, arg2) werden genauso codiert, wie die Elemente des S-Stack.

Aufbau der Closures:

- Alle Closure-Strukturen liegen in einem separaten Speicherbereich, dem Heap (H).

  Hinweis: Es ist nicht erforderlich, Heap-Bereiche wiederzuverwenden, d. h. einmal im Heap abgelegte Daten verbleiben dort bis zum Ende der Berechnung. Desweiteren muß ein Überlauf des Heap nicht unbedingt abgefangen werden.
- Jede Closure besteht aus einem Zeiger auf den zugehörigen Funktionsrumpf, gefolgt von einer Kopie(!) des Environments, das die Instanziierungen der relativ freien Variablen der Funktion enthält.

Implementierung der SECDH-Operationen:

- Ein Maschinenzustand setzt sich aus fünf Zeigern zusammen — nämlich je einem Zeiger auf die Stacks S, E, D, den Code-Bereich C, und den Heap H.
- Für jeden Op-Code `xxxx` (z. B. `PUSH_S`) ist eine Assembler-Funktion `I_xxxx` (für das Beispiel: `I_PUSH_S`) zu definieren, die in den Registern A2 bis A6 einen Maschinenzustand als Argument erwartet und in den selben Registern einen entsprechend modifizierten Maschinenzustand zurückliefert.
- Der Zusammenhang zwischen Op-Code `xxxx` und Adresse `I_xxxx` der aufzurufenen Funktion soll durch eine Op-Code-Tabelle hergestellt werden, wobei der durch den Offset `xxxx` bezeichnete Tabelleneintrag gerade die Adresse `I_xxxx` enthalten soll. (Damit das funktioniert, müssen alle Op-Codes nach Long-Werten ausgerichtetet sein, also Vielfache von 4 sein.)

  Dazu ist ein Speicherbereich anzulegen, der genauso viele Einträge enthält, wie Op-Codes vorhanden sind (`DS.L` Anweisung). Ferner ist eine Funktion `INIT_OP_TAB` zu definieren, die diesen Speicherbereich mit den korrekten Adressen füllt.
- Schreiben Sie eine Funktion `EXEC`, die die Interpretation einer Sequenz von SECDH-Instruktionen vornimmt, d. h.
  - als Argument auf dem Laufzeit-Stack einen Zeiger auf den zu interpretierenden Code erwartet,

– daraus in den Registern A2 bis A6 den entsprechenden Anfangszustand der SECDH-Maschine generiert,
– mit Hilfe der Op-Code-Tabelle die im übergebenen Code vorgefundenen Instruktionen in entsprechende Funktionsaufrufe umsetzt, bis der Op-Code `DONE` gelesen wird,
– das Resultat der Code-Interpretation, also den obersten Eintrag des S-Stack, auf dem Laufzeit-Stack zurückliefert.

Hinweis: Auf der Homepage der Vorlesung finden Sie eine Datei `secdh.template`, die als Gerüst einer Implementierung des SECDH-Interpreters dienen kann.

Aufgabe 21 (10 Punkte)

Es soll ein schrittweises Debugging von SECDH-2 Programmen ermöglicht werden, wobei der Interpreter seinen aktuellen Zustand ausgibt und erst auf Tastendruck den nachfolgenden Zustand erzeugt.

Um eine möglicht aussagekräftige Ausgabe der Stack-Inhalte (insbesondere des S-Stack) zu erreichen, werden sogenannte Tags eingeführt, anhand derer erkennbar ist, um was für eine Art von Objekt es sich bei einem Stack-Eintrag überhaupt handelt. Diese Tags sollen in die beiden niederwertigsten Bits eines jeden Stack-Eintrags wie folgt codiert werden: Bei einem Eintrag mit den niederwertigsten Bits

- `00` handelt es sich um einen Zeiger auf einen Environment-Frame oder in den Programm-Code.
- `01` handelt es sich um eine vorzeichenbehaftete 30-Bit Zahl.
- `10` handelt es sich um einen Zeiger auf eine Closure.
- `11` handelt es sich um einen Bool'schen Wert.

Hinweis: Dies erfordert, daß alle Zeiger eine feste Ausrichtung bezüglich der beiden niederwertigsten Bits haben.

Entsprechend codierte Ausdrücke können mit Hilfe der in der Assembler-Bibliothek `io-pack.ss` zur Verfügung gestellten Funktion `PUT_OBJ` ausgegeben werden. Darüber hinaus stellt diese Bibliothek Funktionen wie z. B. `PUT_ENV` zum Ausgeben ganzer Environment-Ketten oder `PR_STATE` zum Ausgeben eines Maschinenzustandes(!) bereit.

Fügen Sie obige Tags in Ihren Interpreter ein und stellen Sie eine zusätzliche, auf `EXEC` basierende Funktion `EXEC_IO` zur Verfügung, die während der Interpretation alle Zustände schrittweise ausgibt.

Aufgabe 22 (20 Punkte)

Compilieren Sie die folgenden Programme von Hand und testen Sie damit Ihren Interpreter:

a) Roller-Coaster aus der Vorlesung:

```
letrec
  f = lambda u v
           letrec
             g = lambda w z
                   (if (GT w u)
                       (g (- u 1) z)
                       (f v (+ w 1)))
           in (g v u)
in (f 1 2)                          .
```

Zur Klarstellung: `(- 1 2)` compiliert sich zu `PUSH_S 2; PUSH_S 1; MINUS` und ergibt zur Laufzeit den Wert (-1).

b) Erstes Testprogramm für Closures:

```
letrec
  k = lambda x lambda y x
in ((k 2) 3)                        .
```

c) Zweites Testprogramm für Closures:

```
letrec
  k   = lambda x lambda y x
  ap3 = lambda f (f 3)
in (ap3 (k 2))                      .
```

d) Fakultät mit Y-Kombinator (siehe Aufgabe 10).

Abgabe: Di., 02.07.2002 in den Übungen.