ÜBUNGSBLATT 11 — MITTWOCH

Aufgabe 1 (AGS 15.17 b, 15.1 a)

(a) Gegeben sei folgender AM_1-Code:

```
1:  INIT 1;            10:  MUL;               19:  READ(global,1);
2:  CALL 18;           11:  STOREI(-3);        20:  LOADA(global,1);
3:  INIT 0;            12:  LOAD(lokal,-2);    21:  PUSH;
4:  LOAD(lokal,-2);    13:  LIT 1;             22:  LOAD(global,1);
5:  LIT 0;             14:  SUB;               23:  PUSH;
6:  GT;                15:  STORE(lokal,-2);   24:  CALL 3;
7:  JMC 17;            16:  JMP 4;             25:  WRITE(global,1);
8:  LIT 2;             17:  RET 2;             26:  JMP 0;
9:  LOADI(-3);         18:  INIT 0;
```

Dokumentieren Sie 12 Schritte der AM_1 mit der Startkonfiguration $\sigma = (22, \varepsilon, 1:3:0:1,3,\varepsilon,\varepsilon)$.

(BZ,		DK,	Laufzeitkeller,	REF,	Inp,	Out)
22		ε	1 : 3 : 0 : 1	3	ε	ε
23		1	1 : 3 : 0 : 1	3	ε	ε
24		ε	1 : 3 : 0 : 1 : 1	3	ε	ε
3		ε	1 : 3 : 0 : 1 : 1 : 25 : 3	7	ε	ε
4		ε	1 : 3 : 0 : 1 : 1 : 25 : 3	7	ε	ε
5		1	1 : 3 : 0 : 1 : 1 : 25 : 3	7	ε	ε
6	1 > 0 ?	0 : 1	1 : 3 : 0 : 1 : 1 : 25 : 3	7	ε	ε
7		1	1 : 3 : 0 : 1 : 1 : 25 : 3	7	ε	ε
8		ε	1 : 3 : 0 : 1 : 1 : 25 : 3	7	ε	ε
9		2	1 : 3 : 0 : 1 : 1 : 25 : 3	7	ε	ε
10		1 : 2	1 : 3 : 0 : 1 : 1 : 25 : 3	7	ε	ε
11		2	1 : 3 : 0 : 1 : 1 : 25 : 3	7	ε	ε
12		ε	2 : 3 : 0 : 1 : 1 : 25 : 3	7	ε	ε
13		1	2 : 3 : 0 : 1 : 1 : 25 : 3	7	ε	ε

(b) Gegeben sei folgendes C_1-Programm:

```
1   #include <stdio.h>
2   int b;
3   void f(int a, int *b) {
4     int c;
5     c = a;
6     while (c > 0) {
7       *b = *b * 2;
8       c = c - 1;
9     }
10  }
11
12  void main() {
13    int a;
14    scanf("%i", &a);
15    b = 1;
16    f(a, &b);
17    printf("%d", b);
18  }
```

1 — 1.1 — 1.2

Übersetzen Sie das Programm mittels *trans* in ein AM_1-Programm mit Baumstrukturierten Adressen. Geben Sie zunächst die Symboltabellen tab_{main} und tab_f zur Übersetzung der Statements in den Funktionen `main` bzw. `f` mittels *stseqtrans* an. Geben Sie keine weiteren Zwischenschritte an.

activation record für f:

a	b	ra	par	c
-3	-2	-1	0	1

↑ REF

tab_f = [f / (proc, 1), a / (var, lokal, -3), b / (var-ref, -2), c / (var, lokal, 1)]

tab_{main} = [f / (proc, 1), b / (var, global, 1), a / (var, lokal, 1)]

```
INIT 1;        (Speicher für glob. Variable)
CALL 2;        (Aufruf der Funktion f)
JMP  0;        (Ende des Programms)
```

// Übersetzung von f

```
1:    INIT 1;          (Speicher für lok. Variable)
      LOAD (lokal, -3);   } c = a;
      STORE (lokal, 1);   }
1.2.1: LOAD (lokal, 1);   }
      LIT 0;              } c > 0
      GT;                 }
      JMC 1.2.2;
      LOADI (-2);         }
      LIT 2;              } *b = *b * 2;
      MUL;                }
      STOREI (-2);        }
      LOAD (lokal, 1);    }
      LIT 1;              } c = c - 1;
      SUB;                }
      STORE (lokal, 1);   }
      JMP 1.2.1;
1.2.2: RET 2;        (Speicher der zwei Parameter freigeben)
```

// Übersetzung von main

```
2:    INIT 1;
      READ (lokal, 1);    } scanf(...)
      LIT 1;              } b = 1;
      STORE (global, 1);  }
      LOAD (lokal, 1);    }
      PUSH;               }
      LOADA (global, 1);  } f(a, &b);
      PUSH                }
      CALL 1;             }
      WRITE (global, 1);  } printf(...)
      RET 0;
```

Aufgabe 2 (AGS 15.16 a, AGS 15.18 b)

(a) Gegeben sei folgendes Fragment eines C_1-Programms:

```
1  #include <stdio.h>
2
3  int x, y;
4
5  void f(...) {...}
6  void main() {...}
7  void g(int a, int *b) {
8    int c;
9    c = 3;
10   if (c == *b)
11     while (a > 0) f(&a, b);
12 }
```

kein INIT und RET benötigt

Übersetzen Sie die Sequenz der Statements im Rumpf von g in entsprechenden AM_1-Code mit baumstrukturierten Adressen (mittels *stseqtrans*). Sie brauchen keine Zwischenschritte anzugeben. Geben Sie zunächst die benötigte Symboltabelle tab_g an.

tab_g = [f / (proc, 1), g / (proc, 2),
x / (var, global, 1), y / (var, global, 2),
a / (var, lokal, -3), b / (var-ref, -2), c / (var, lokal, 1)]

```
LIT 3;
STORE (lokal, 1);

LOAD (lokal, 1);
LOADI (-2);
EQ;
JMC 2.2.1;
```

} if-Bed.

Frt. g ↑ 2.Statement in g

```
2.2.2.1: LOAD (lokal, -3);
         LIT 0;
         GT;
         JMC 2.2.2.2
```

} Schleifenbed.

then-Zweig ↑ Schleifenbed. ↑

```
         LOADA (lokal, -3);
         PUSH;
         LOAD (lokal, -2);
         PUSH;
         CALL 1;
         JMP 2.2.2.1;
```

} Schleifenrumpf

```
2.2.2.2:
2.2.1:
```

(b) Gegeben sei folgender AM_1-Code:

```
1:  INIT 1;            8:  LOADI(-2);        15: LOADA(global, 1);
2:  CALL 13;           9:  LIT 2;            16: PUSH;
3:  INIT 0;           10:  DIV;              17: CALL 3;
4:  LOADI(-2);        11:  STOREI(-2);       18: WRITE(global, 1);
5:  LIT 2;            12:  RET 1;            19: JMP 0;
6:  GT;               13:  INIT 0;
7:  JMC 12;           14:  READ(global, 1);
```

Erstellen Sie ein Ablaufprotokoll der AM_1, indem Sie sie schrittweise ablaufen lassen, bis die Maschine terminiert. Die Anfangskonfiguration sei $(14, \varepsilon, 0 : 0 : 1, 3, 4, \varepsilon)$. Sie müssen nur Zellen ausfüllen, deren Wert sich im Vergleich zur letzten Zeile geändert hat.

(BZ,	DK,	Laufzeitkeller,	REF,	Inp,	Out)
(14,	ε,	0 : 0 : 1 ,	3,	4,	ε)
(15,	ε,	4 : 0 : 1 ,	3,	ε,	ε)
(16,	1,	4 : 0 : 1 ,	3,	ε,	ε)
(17,	ε,	4 : 0 : 1 : 1,	3,	ε,	ε)
(3	ε,	4 : 0 : 1 : 1 : 18 : 3 ,	6,	ε,	ε)
(4,	ε,	4 : 0 : 1 : 1 : 18 : 3 ,	6,	ε,	ε)
(5,	4, ←	4 : 0 : 1 : 1 : 18 : 3 ,	6 ,	ε,	ε)
(6 ,	2 : 4,	4 : 0 : 1 : 1 : 18 : 3 ,	6 ,	ε,	ε)
(7,	4 > 2? 1,	4 : 0 : 1 : 1 : 18 : 3,	6,	ε,	ε)
(8,	ε,	4 : 0 : 1 : 1 : 18 : 3,	6,	ε,	ε)
(9,	4 , ←	4 : 0 : 1 : 1 : 18 : 3,	6,	ε,	ε)
(10,	2 : 4 ,	4 : 0 : 1 : 1 : 18 : 3,	6,	ε,	ε)
(11 ,	4/2 = 2 ,	4 : 0 : 1 : 1 : 18 : 3 ,	6,	ε,	ε)
(12 ,	ε,	2 : 0 : 1 : 1 : 18 : 3,	6 ,	ε,	ε)
(18,	ε,	2 : 0 : 1 ,	3,	ε,	ε)
(19,	ε,	2 : 0 : 1 ,	3,	ε,	2)
(0,	ε,	2 : 0 : 1 ,	3,	ε,	2)