<u>Musterlösung:</u> DIV (Magister) 4. Serie

1.) 5 Pkt. Ein-Adress-Rechner für Berechnung f(n) = n! (1 Pkt. Protokoll)

```
.START 100 ;Einsprungadresse
1       3               ; n
2       1               ; f(n), vorbelegt mit 1
10      1               ; eine 1
100     =>      0       ; RR 0
101     +       1       ; n laden
102     S>      104     ; n>0 ? weiter : halt
103     H       0       ; Halt
104     =>      2       ; in f(n) speichern
105     +       1       ; n laden
106     -       10      ; n-1
107     ->      1       ; neues n ohne Löschen in n speichern
108     S>      110     ; n>0 ? weiter : halt
109     H       0       ; Programmende
110     *       2       ; n * f(n)
111     =>      2       ; neues f(n) speichern
112     S       105     ; Schleife
```

SIMNEU-Protokoll

```
;    oc   { op ,   addr }      rr    zr     1     2     10
;   100   { =>,     0 }       0     0     3     1     1
;   101   { +,      1 }       0     0     3     1     1
;   102   { S>,    104 }      3     1     3     1     1
;   104   { =>,     2 }       3     1     3     3     1
;   105   { +,      1 }       0     1     3     3     1
;   106   { -,     10 }       3     1     3     3     1
;   107   { ->,     1 }       2     1     2     3     1
;   108   { S>,    110 }      2     1     2     3     1
;   110   { *,      2 }       2     1     2     3     1
;   111   { =>,     2 }       6     1     2     6     1
;   112   { S,    105 }       0     1     2     6     1
;   105   { +,      1 }       0     1     2     6     1
;   106   { -,     10 }       2     1     2     6     1
;   107   { ->,     1 }       1     1     1     6     1
;   108   { S>,    110 }      1     1     1     6     1
;   110   { *,      2 }       1     1     1     6     1
;   111   { =>,     2 }       6     1     1     6     1
;   112   { S,    105 }       0     1     1     6     1
;   105   { +,      1 }       0     1     1     6     1
;   106   { -,     10 }       1     1     1     6     1
;   107   { ->,     1 }       0     0     0     6     1
;   108   { S>,    110 }      0     0     0     6     1
;   109   { H,      0 }       0     0     0     6     1
```

2.) 5 Pkt. Ein-Adress-Rechner Fibonaccizahlen f(n+1) = f(n-1) + f(n) AB: f(1)=1 ; f(2)=1

f (n) = f (n – 2) + f (n – 1)

```
.START 101
1       1               ; f(n-2), vorbelegt mit f(1)=1
2       1               ; f(n-1), vorbelegt mit f(2)=1
3       0               ; f(n) - Ergebnis
10      5               ; n, hier 5
20      1               ; eine Eins
101     =>      0       ; RR 0
102     +       10      ; n laden
103     -       20      ; n--
;Schleifenbeginn
104     -       20      ; n--
105     ->      10      ; n speichern ohne Löschen
106     S>      108     ; n>0 ? weiter : halt
107     H       0       ; Programmende
108     =>      0       ; RR 0
109     +       1       ; f(n-2) laden
110     +       2       ; f(n-1) addieren
111     =>      3       ; in f(n) abspeichern
112     +       2       ; f(n-1) laden
113     =>      1       ; als f(n-2) abspeichern
114     +       3       ; f(n) laden
115     =>      2       ; als f(n-1) abspeichern
116     +       10      ; n laden
117     S       104     ; zum Schleifenbeginn
```

SIMNEU-Protokoll

```
;    oc  {   op , addr }        rr    zr     1     2     3    10    20
;   101  { =>,     0 }         0     0     1     1     0     5     1
;   102  { +,     10 }         0     0     1     1     0     5     1
;   103  { -,     20 }         5     1     1     1     0     5     1
;   104  { -,     20 }         4     1     1     1     0     5     1
;   105  { ->,    10 }         3     1     1     1     0     3     1
;   106  { S>,   108 }         3     1     1     1     0     3     1
;   108  { =>,     0 }         3     1     1     1     0     3     1
;   109  { +,      1 }         0     1     1     1     0     3     1
;   110  { +,      2 }         1     1     1     1     0     3     1
;   111  { =>,     3 }         2     1     1     1     2     3     1
;   112  { +,      2 }         0     1     1     1     2     3     1
;   113  { =>,     1 }         1     1     1     1     2     3     1
;   114  { +,      3 }         0     1     1     1     2     3     1
;   115  { =>,     2 }         2     1     1     2     2     3     1
;   116  { +,     10 }         0     1     1     2     2     3     1
;   117  { S,     104 }        3     1     1     2     2     3     1
;   104  { -,     20 }         3     1     1     2     2     3     1
;   105  { ->,    10 }         2     1     1     2     2     2     1
;   106  { S>,   108 }         2     1     1     2     2     2     1
;   108  { =>,     0 }         2     1     1     2     2     2     1
;   109  { +,      1 }         0     1     1     2     2     2     1
;   110  { +,      2 }         1     1     1     2     2     2     1
;   111  { =>,     3 }         3     1     1     2     3     2     1
;   112  { +,      2 }         0     1     1     2     3     2     1
;   113  { =>,     1 }         2     1     2     2     3     2     1
;   114  { +,      3 }         0     1     2     2     3     2     1
;   115  { =>,     2 }         3     1     2     3     3     2     1
;   116  { +,     10 }         0     1     2     3     3     2     1
```

```
;   117  { S,      104 }          2      1      2      3      3      2      1
;   104  { -,       20 }          2      1      2      3      3      2      1
;   105  { ->,      10 }          1      1      2      3      3      1      1
;   106  { S>,     108 }          1      1      2      3      3      1      1
;   108  { =>,       0 }          1      1      2      3      3      1      1
;   109  { +,        1 }          0      1      2      3      3      1      1
;   110  { +,        2 }          2      1      2      3      3      1      1
;   111  { =>,       3 }          5      1      2      3      5      1      1
;   112  { +,        2 }          0      1      2      3      5      1      1
;   113  { =>,       1 }          3      1      3      3      5      1      1
;   114  { +,        3 }          0      1      3      3      5      1      1
;   115  { =>,       2 }          5      1      3      5      5      1      1
;   116  { +,       10 }          0      1      3      5      5      1      1
;   117  { S,      104 }          1      1      3      5      5      1      1
;   104  { -,       20 }          1      1      3      5      5      1      1
;   105  { ->,      10 }          0      0      3      5      5      0      1
;   106  { S>,     108 }          0      0      3      5      5      0      1
;   107  { H,        0 }          0      0      3      5      5      0      1
```

3.) Ein-Adress-Rechner, der die Reihenfolge der Elemente eines Feldes a der Länge n invertiert.

```
.START 1000
1       6               ; Feldgröße n
2       11              ; Startadresse des Feldes a
11      1               ; Testdatenfeld
12      3
13      4
14      5
15      6
16      7
100     16              ; für Kodierung
101     1               ; Code von '+'
102     6               ; Code von '=>'
103     0               ; Zwischenspeicher für den Wert des Startfeldes
104     0               ; aktuelles Feld
105     1               ; eine Eins
1000    =>      0       ; RR 0
1001    +       1       ; n laden
1002    -       105     ; n--
1003    ->      1       ; n speichern ohne Löschen
1004    S>      1006    ; n>0 ? weiter : halt
1005    H       0       ; Halt
1006    +       2       ; n + Startadresse
1007    =>      104     ; als aktuelles Feld a[n] speichern
1008    S       2000    ; Sprung zur Codeerzeugung
1009    +       1       ; n laden
1010    -       105     ; n--
1011    =>      1       ; n speichern
1012    +       2       ; Startadresse laden
1013    +       105     ; um eins erhöhen
1014    =>      2       ; neue Adresse speichern
1015    S       1001    ; Schleife
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Unterprogramm - Codeerzeugung:
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
2000  +   2     ; aktuelle Startadresse laden
2001  *   100   ; 16 * AT
2002  +   101   ; + OT
2003  =>  3000  ; Code in 3000 speichern
2004  +   104   ; aktuelles a[n] laden
2005  *   100   ; 16 * AT
2006  +   101   ; + OT
2007  =>  3002  ; Code in 3002 speichern
2008  +   2     ; aktuelle Startadresse laden
2009  *   100   ; 16 * AT
2010  +   102   ; + OT
2011  =>  3003  ; Code in 3003 speichern
2012  +   104   ; aktuelles a[n] laden
2013  *   100   ; 16 * AT
2014  +   102   ; + OT
2015  =>  3005  ; Code in 3005 speichern
2016  S   3000  ; Erzeugtes Unterprogramm aufrufen
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Unterprogramm zum Tauschen der Feldinhalte;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
3000  H   0     ; wird von 2003 überschrieben - Wert der Startadresse laden
3001  =>  103   ; in Zwischenspeicher
3002  H   0     ; wird von 2007 überschrieben - Wert der Adresse a[n] laden
3003  H   0     ; wird von 2011 überschreiben - speichern auf Startadresse
3004  +   103   ; Zwischenspeicher laden
3005  H   0     ; wird von 2015 überschreiben - speichern auf a[n]
3006  S   1009  ; Sprung zurück ins Hauptprogramm
```

4.) a) **2 Pkt. Grammatik** G = (T , N , P , <S>) mit T = { a , b } ; N = { <S> , <A> , <B>}
P: (siehe Aufgabenzettel)
alle Wörter mit höchstens 4-Buchstaben durch Ableitungsbaum/-kette aufstellen:
<S> -> a<B> -> ab ;   <S> -> b<A> -> ba
<S> -> a<B> -> aa<B><B> ->* aabb
<S> -> b<A> -> bb<A><A> ->* bbaa
<S> -> a<B> -> ab<S> -> aba<B> -> abab
<S> -> b<A> -> ba<S> -> bab<A> -> baba
<S> -> a<B> -> ab<S> -> abb<A> -> abba
<S> -> b<A> -> ba<S> -> baa<B> -> baab
mögliche Wörter: *ab , ba , abab , abba , aabb , baab , baba , bbaa*

b) **4 Pkt.**(vollst. Induktion bzw. Begründung (2 Pkt.) für #a = #b für alle Wörter von L(G))
IA: ab, ba -> #a(ab) = #b(ab); #a(ba) = #b(ba) bei 2 Ableitungen
IV: w ∈ L(G) -> #a(w) = #b(w), Anz. der Ableitungen für w ist n <u>(n>2 & n gerade)</u>
IB: v ∈ L(G), Anz. der Ableitungen für v ist n+2 *(bei n+1 ist v ∉ L(G))*
v enthält nur Nichtterminalsymbole und kann nur so entstanden sein:
v ≅ w<S> ≅ wa<B> oder wb<A> nach IV stimmt Behauptung
v ≅ u<A><A>, v ≅ u<B><B> erfordert weitere Betrachtung
um <A><A> abzuleiten muß vorher *ein <A>* zu b<A><A> abgeleitet werden
dieses <A> kann aus <S> -> b<*A*> oder <A> -> b<A><*A*> abgeleitet werden
keine andere Möglichkeit existiert -> so folgt: u=w<S> oder u=wa<A>|b<A>,
so folgt mit IV: #a(v)=#b(v), analog v ≅ u<B><B>