Universität Mannheim
Fakultät für Mathematik und Informatik
Lehrstuhl für Praktische Informatik IV
Professor Dr. W. Effelsberg

Scheinklausur Informatik
SS 2000: Praktische Informatik II

Name: ..... Vorname: .....

Matrikel-Nr.: ..... Semester: ..... Fach: .....

Klausurergebnis per SMS (Telefonnummer): ..... (optional)

Hinweise:

1. Bitte füllen Sie sofort den Kopf des Deckblattes aus.
2. Überprüfen Sie bitte Ihr Klausurexemplar auf Vollständigkeit (**19** Seiten).
3. Tragen Sie die Lösungen – soweit möglich – direkt in die Klausur ein.
4. Zugelassene Hilfsmittel: nicht programmierbarer Taschenrechner
5. Bearbeitungszeit: 90 Minuten.

Aufgabe	max. Punktzahl	Punkte
1	10	
2	15	
3	18	
4	20	
5	15	
6	12	
Summe	90	

Aufgabe 1: Schaltnetze *[4+6=10 Punkte]*

Gegeben sei die Funktion $F(x_0, x_1, x_2, x_3)$ durch folgende Wahrheitstabelle. *Don't care*-Felder sind dabei mit X gekennzeichnet.

x_0	x_1	x_2	x_3	F
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	X
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	X
1	1	1	0	0
1	1	1	1	1

(a) [4 Punkte] Tragen Sie die Werte von F in untenstehendes Karnaugh-Diagramm ein.

$\frac{x_2x_3}{x_0x_1}$	00	01	11	10
00				
01				
11				
10				

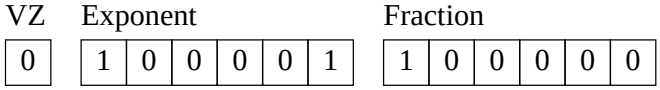
Hinweise zur Korrektur:

- Für jeden falsch eingetragenen Wert 0,5 Punkt Abzug

(b) [6 Punkte] Wie lauten die Primimplikanten und die minimale disjunktive Form der Funktion F? Markieren Sie die Blöcke, die Sie zusammengefaßt haben, im Karnaugh-Diagramm.

Aufgabe 2: Zahlendarstellung *[4+8+3=15 Punkte]*

Das folgende Format für Fließkommazahlen sei gegeben:



- Vorzeichen (1 bit): $0 \equiv$ positiv, $1 \equiv$ negativ
- Fraction (6 bit): Normalisierung $1 \leq 1,\text{Fraction} < 2$, hidden bit
- Exponent (6 bit): Excess-Darstellung (Excess = $2^{6-1} = 32$)

(a) [4 Punkte] Gegeben sei die Bitfolge $b = 0\ 100001\ 100000_2$ in obigem Fließkommaformat (d.h. VZ = 0_2, Exponent = 100001_2, Fraction = 100000_2). Wandeln Sie b in eine Dezimalzahl um.

(b) [8 Punkte] Wandeln Sie die beiden Dezimalzahlen 129 und $1,2$ in obiges Fließkommaformat um. Lassen sich die beiden Zahlen exakt in diesem Format darstellen? Geben Sie gegebenenfalls den durch die Darstellung entstehenden absoluten Fehler an.

(c) [3 Punkte] Entwickeln Sie einen Vorschlag zur Repräsentation des Wertes 0 in obiger Fließkommadarstellung. Welche Auswirkungen hat Ihr Vorschlag auf den darstellbaren Wertebereich?

Aufgabe 3: Mikroprogrammierung *[12+6=18 Punkte]*

Betrachten Sie den aus der Vorlesung bekannten mikroprogrammgesteuerten Computer (**siehe Anlage auf Seite 19**). Dazu sei folgendes Mikroprogramm gegeben:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
0:			1										1				1					1
1:		1				1	1		1								1					1
2:																			1			1
3:							1		1	1								1				
4:																				1		1
5:							1											1				
6:	1									1							1					1
7:																		1				
8:		1											1				1					1
9:			1			1	1				1						1					1
10:																		1				
11:			1										1				1					1
12:														1		1	1					1

(a) [12 Punkte] Schreiben Sie das Mikroprogramm in die symbolische Darstellung um. Kommentieren Sie die einzelnen Schritte. Hinweis: Denken Sie an die besondere Bedeutung des Steuersignals 18 (Instruktionen 3,5,7,10).

(b) [6 Punkte] Führen Sie das Programm mit den Startwerten $B = 30$ und $C = 20$ aus. Dokumentieren Sie die einzelnen Schritte der Programmausführung wie folgt: Geben Sie jeweils nach Abarbeitung der Instruktion 1 den Wert von Register *A*, nach Abarbeitung der Instruktion 6 den Wert von Register *B* und nach Abarbeitung der Instruktion 9 den Wert von Register *C* an. Welchen Wert enthält das Register *MDR* am Ende des Programms (Instruktion 12)?

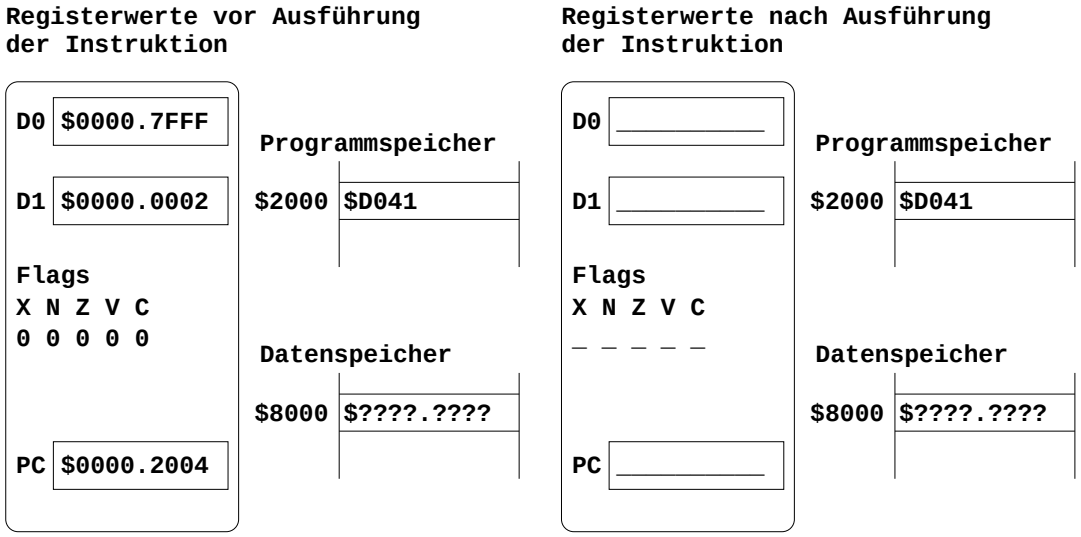
Aufgabe 4: Assemblerprogrammierung *[10+10=20 Punkte]*

Hinweis: Auszüge aus dem Befehlssatz des M68000 finden Sie auf Seite 18.

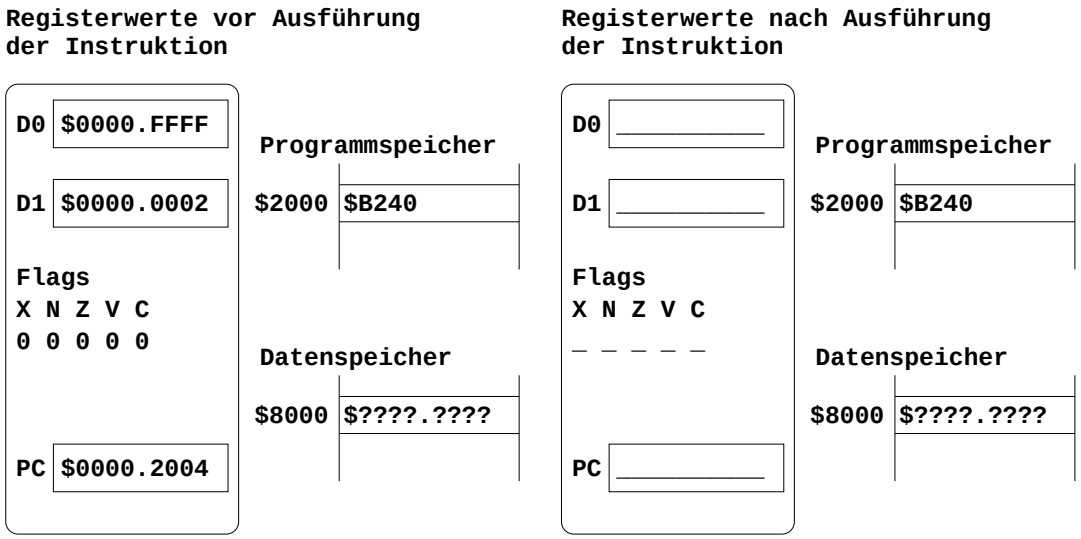
(a) [10 Punkte] Adressierungsarten

Geben Sie den Inhalt aller mit "_" markierten Register, Flags und Speicherstellen nach Ausführung der Assemblerinstruktion an:

ADD.W D1,D0

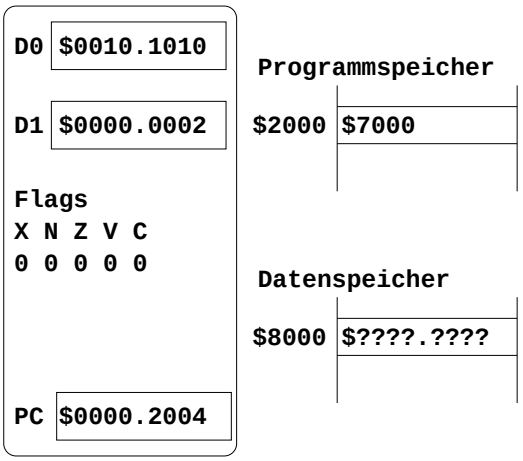


CMP.W D1,D0

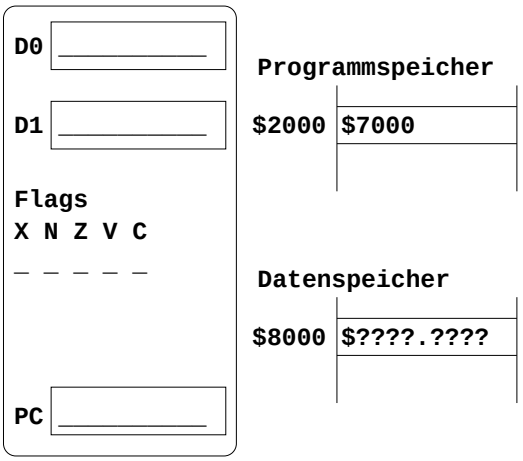


MOVEQ #$00,D0

Registerwerte vor Ausführung der Instruktion

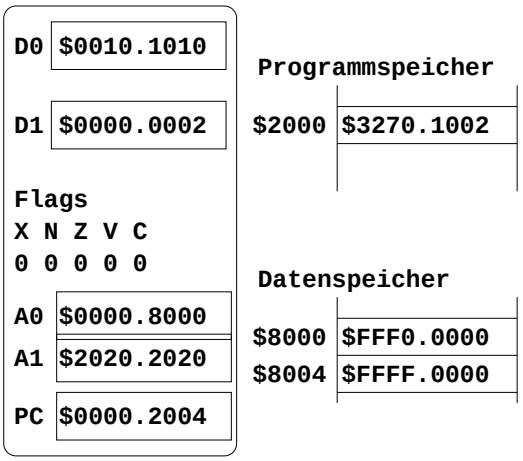


Registerwerte nach Ausführung der Instruktion

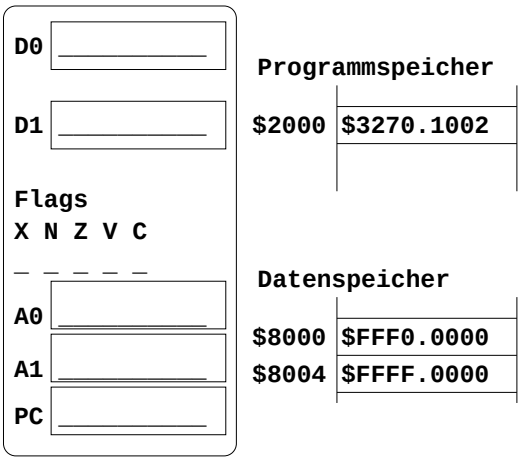


MOVE.W $2(A0,D1),A1

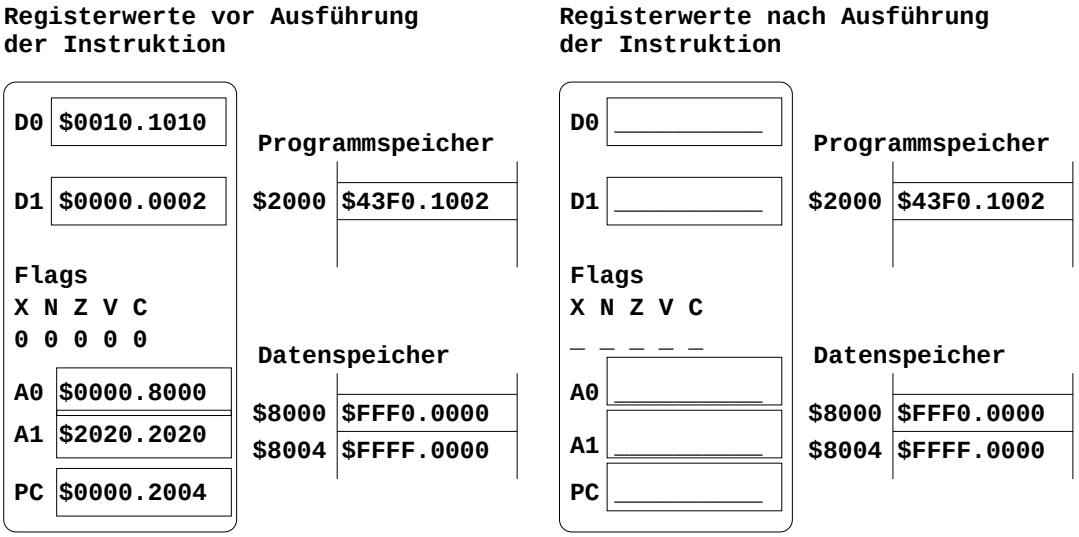
Registerwerte vor Ausführung der Instruktion



Registerwerte nach Ausführung der Instruktion



LEA $2(A0,D1),A0


Registerwerte vor Ausführung der Instruktion
D0 $0010.1010
D1 $0000.0002
Flags
X N Z V C
0 0 0 0 0
A0 $0000.8000
A1 $2020.2020
PC $0000.2004
Programmspeicher
$2000 $43F0.1002
Datenspeicher
$8000 $FFF0.0000
$8004 $FFFF.0000
Registerwerte nach Ausführung der Instruktion
D0
D1
Flags
X N Z V C
A0
A1
PC
Programmspeicher
$2000 $43F0.1002
Datenspeicher
$8000 $FFF0.0000
$8004 $FFFF.0000

(b) [10 Punkte] Addressierungsarten und höhere Programmiersprachen

Gegeben sei folgendes C-Programm:

```
001   struct A
002   {
003     int b;
004     int c;
005   };
006
007   int main()
008   {
009     register int i;
010     static struct A a[100];
011
012     for (i= 0; i < 100 ; i++)
013       a[i].c= 0;
014
015     return 0;
016   }
```

Übersetzen Sie das Programm in Assembler-Code. Bilden Sie dabei den Elementzugriff in Zeile `013` möglichst exakt auf eine Addressierungsart des Motorola 68000 ab. Verwenden Sie den auf der folgenden Seite abgedruckten Programmkopf als Vorlage.

Hinweise: Gehen Sie dabei davon aus, daß Integer–Variablen 16–Bit lang sind. Das Schlüsselwort `register` weist den Compiler an, diese Variable in einem CPU-Register zu speichern. Das Schlüsselwort `static` weist den Compiler an, das Array nicht auf dem Stack anzulegen, sondern im Datenbereich des Hauptspeichers.

```
        ORG     $0
        DC.L    $8000    Wert des Stack Pointers nach einem reset
        DC.L    MAIN     Wert des Program Counters nach einem reset

        ORG     $2000    Start at location 2000 Hex

MAIN    MOVE.W  #$0,D1
```

Aufgabe 5: Assemblerprogrammierung *[7+4+4=15 Punkte]*

Nachfolgend ist das Listing eines Programmes in Motorala 68000 Assembler abgedruckt.

Hinweis: Auszüge aus dem Befehlssatz des M68000 finden Sie auf Seite 18.

```
        ORG     $0
        DC.L    $8004           Stack pointer value after a reset
        DC.L    START           Program counter value after a reset

        ORG     $2000           Start at location 2000 Hex

START   JSR     MAIN
        BREAK

MAIN    MOVEQ   #$05,D0
        JSR     WAS
        RTS

WAS     MOVE.L  D0,-(SP)
        MOVE.L  D7,-(SP)
        MOVE.L  D0,D7
        MOVEQ   #$1,D0
        CMP.L   D7,D0
        BGE.S   M0              ; BGE = branch on greater or equal
        MOVE.L  D7,D0
        ASR.L   #$1,D0          ; ASR = arithmetic shift right (D0 = D0/2)
        JSR     WAS
M0      MOVEQ   #$1,D0
LABEL0  AND.L   D7,D0
        JSR     FPRINT          ; gibt die Zahl in D0 als ASCII aus
        MOVE.L  (SP)+,D7
        MOVE.L  (SP)+,D0
        RTS
```

(a) [7 Punkte] Zeichnen Sie den Stackinhalt bis zum Programmlabel `LABEL0` in untenstehende Tabelle. Beginnen Sie im Hauptprogramm am Label `main`.

Adresse	Wert
7FD0	
7FD4	
7FD8	
7FDC	
7FE0	
7FE4	
7FE8	
7FEC	
7FF0	
7FF4	
7FF8	
7FFC	
8000	RETURN ADRESS (JSR MAIN)

(b) [4 Punkte] Fertigen Sie eine Tabelle, aus der hervorgeht, welche Zahlen in welcher Reihenfolge auf dem Bildschirm ausgegeben werden.

(c) [4 Punkte] Was leistet das Programm im Allgemeinen?

Aufgabe 6: Lexikalische Analyse *[10+2=12 Punkte]*

(a) [10 Punkte] Gegeben seien die folgenden Produktionsregeln für eine `for`-Schleife als Ausschnitt aus einer größeren Menge von Produktionsregeln:

```
For-Anweisung    -> for (For-Init; Bedingung; For-Update)
                        Zuweisung;
For-Init         -> Lokale-Variable
Bedingung        -> Name Relation Nummer
For-Update       -> Zuweisung

Lokale-Variable -> Typ Name = Nummer
Typ              -> int | short

Zuweisung        -> Name = Ausdruck
Ausdruck         -> Name Operator Name | Name Operator Nummer
Operator         -> + | -
Relation         -> > | < | == | !=
```

Ein `Name` besteht dabei aus einer beliebigen Buchstabefolge und eine `Nummer` aus einer beliebigen ganzen Zahl. Zerlegen Sie damit die folgende Anweisung in Token und bauen Sie einen Zerlegungsbaum auf:

```
for (int i = 0; i < 10; i = i + 1) z = z + i;
```

(b) [2 Punkte] Ändern Sie die obigen Produktionsregeln so ab, daß auch geschachtelte `for`-Schleifen folgender Art möglich sind:

```
for (int i = 0; i < 10; i = i + 1)
    for (int j = 0; j < 10; j = j + 1)
        z = i + j;
```

Anlage: Auszug aus dem M68000-Instruktionssatz

Mnemonic	XNZVC	BWL	Description	Notes
ADD s,d	*****	XXX	Add binary	d=d+s
ADDA s,An	-----	XX	Add Address	An=An+s
ADDI #e,d	*****	XXX	Add Immediate	d=d+e
ADDQ #q,d	*****	XXX	Add Quick	d=d+q
ADDX s,d	*****	XXX	Add Extended	d=d+s+X
AND s,d	-**00	XXX	Logical AND	d=d&s
ANDI #e,d	-**00	XXX	Logical AND Immediate	d=d&e
ASL d	*****	XXX	Arithmetic Shift Left	d=d*2
ASR d	*****	XXX	Arithmetic Shift Right	d=d/2
Bcc l	-----	XX	Branch conditionally	If cc BRA
BRA l	-----	XX	Branch Always	PC=l
BSR l	-----	XX	Branch to Subroutine	-[SP]=PC,PC=l
CHK s,Dn	-*???	X	Check register	If 0>Dn>s $[18H]
CLR d	-0100	XXX	Clear operand	d=0
CMP s,Dn	-****	XXX	Compare	Dn-s
CMPA s,An	-****	XXX	Compare Address	An-s
CMPI #e,d	-****	XXX	Compare Immediate	d-e
CMPM s,d	-****	XXX	Compare Memory	d-s
DBcc Dn,l	-----		Decrement and Branch	If~cc&Dn-1~-1 BRA
DIVS s,Dn	-***0	X	Signed Division	Dn={Dn%s,Dn/s}
DIVU s,Dn	-***0	X	Unsigned Division	Dn={Dn%s,Dn/s}
EOR Dn,d	-**00	XXX	Exclusive OR	d=dxDn
EORI #e,d	-**00	XXX	Exclusive OR Immediate	d=dxe
EXG r,r	-----	X	Exchange registers	r<->r
EXT Dn	-**00	XX	Extend sign	Dn<hi>=Dn<7or15>
JMP d	-----		Jump	PC=d
JSR d	-----		Jump to Subroutine	-[SP]=PC,PC=d
LEA s,An	-----	X	Load Effective Address	An=EA{s}
LINK An,#nn	-----		Link and allocate	-[SP]=An=SP=SP+nn
LSL d	***0*	XXX	Logical Shift Left	d= {C,d,0}<-
LSR d	***0*	XXX	Logical Shift Right	d=->{C,d,0}
MOVE s,d	-**00	XXX	Move data	d=s
MOVEM s,d	-----	XX	Move Multiple register	rr=s or d=rr
MOVEQ #q,d	-**00	X	Move Quick	d=q
MULS s,Dn	-**00	X	Signed Multiply	Dn<0:31>=Dn*s
MULU s,Dn	-**00	X	Unsigned Multiply	Dn<0:31>=Dn*s
NEG d	*****	XXX	Negate	d=-d
NEGX d	*****	XXX	Negate with Extend	d=-d-X
NOP	-----		No Operation	
NOT d	-**00	XXX	Logical NOT	d=~d
OR s,d	-**00	XXX	Inclusive OR	d=dvs
ORI #e,d	-**00	XXX	Inclusive OR Immediate	d=dve
PEA s	-----	X	Push Effective Address	-[SP]=EA{s}
ROL d	-**0*	XXX	Rotate Left	d= {d}<-
ROR d	-**0*	XXX	Rotate Right	d=->{d}
RTS	-----		Return from Subroutine	PC=[SP]+
Scc d	-----	X	Set conditionally	d=0 or d=-1
SUB s,d	*****	XXX	Subtract binary	d=d-s
SUBA s,An	-----	XX	Subtract Address	An=An-s
SUBI #e,d	*****	XXX	Subtract Immediate	d=d-e
SUBQ #q,d	*****	XXX	Subtract Quick	d=d-q
SUBX s,d	*****	XXX	Subtract with Extend	d=d-s-X
SWAP Dn	-**00	X	Swap register halves	Dn<hi><->Dn<lo>
TST d	-**00	XXX	Test	d
UNLK An	-----		Unlink	SP=An,An=[SP]+
cc			Condition = (T/F/HI/LS/CC/CS/NE/EQ/VC/VS/PL/MI/GE/LT/GT/LE)	

Anlage: Mikroprogrammierung