Mathematische Grundlagen

Die Matrizenmultiplikation $C=AB$ ist nur definiert, wenn A genauso viele Spalten wie B Zeilen hat. Das Element c_{ij} der Matrix C ergibt sich als

$$c_{ij} = a_i \cdot b_j = \sum_{k=1}^{n} a_{ik} b_{kj} ,$$

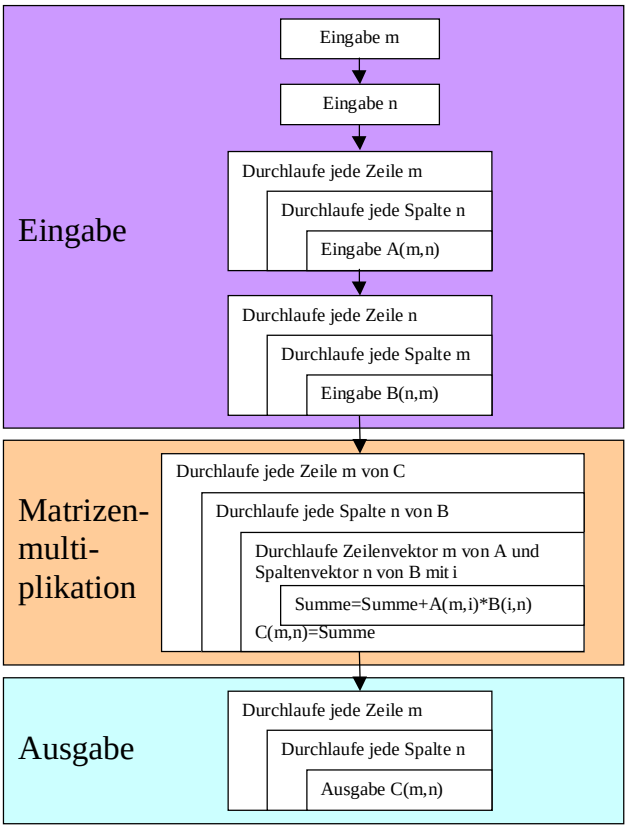
wobei a_i ein Zeilenvektor und b_j ein Spaltenvektor ist. Das Ergebnis ist eine Matrix, die die Dimension (m,m) hat. Für die komplette Matrix C gilt somit:

$$C = \sum_{i=1}^{m} \sum_{j=1}^{n} \sum_{k=1}^{n} a_{ik} b_{kj}$$

Die Multiplikation von Matrizen ist nicht kommutativ, d.h. $AB \neq BA$.

Grober Programmablauf

Das Programm besteht aus Eingabe, Matrizenmultiplikation und Ausgabe:



Strukturierung der Problemlösung

Die Speicherung der Matrizen erfordert insgesamt $2mn+m^2$ Speicherzellen. Der verwendete Datentyp Integer belegt pro Speicherzelle 4 Bytes, deshalb muss bei der Adressierung einer Speicherzelle stets der Faktor 4 beachtet werden.

Die Eingabe der Matrizen A und B unterscheidet sich jeweils durch die Dimension (A hat das Format (m,n), B dagegen (n,m)) und durch die Adresse im Hauptspeicher, wo die Daten abgelegt werden. Da man diese 3 Werte als Parameter problemlos übergeben kann, wird ein Unterprogramm *MatrizenEingabe* dies übernehmen können. Die Eingabe der Dimensionen selbst, die natürlich *vor* der Matrizeneingabe erfolgen muss, ist derart einfach,

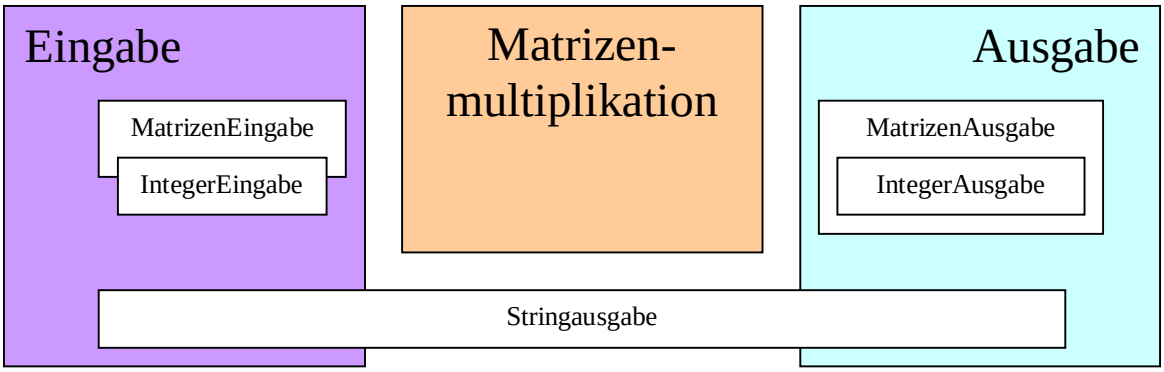
dass ich sie nicht in ein Unterprogramm auszulagern brauche. Die Gesamtheit aller Eingaben wird in *Eingabe* zusammengefaßt.

Die eigentliche Multiplikation von Matrizen erfolgt durch drei verschachtelte Schleifen, die ich zusammen in ein Unterprogramm *Verarbeitung* packe.

Die Ausgabe der Matrix *C* ähnelt wieder sehr stark dem Unterprogramm *MatrizenEingabe*. Trotzdem habe ich ein eigenes Unterprogramm *Matrizenausgabe* geschrieben, insbesondere diente es zu Testzwecken während der Programmierung.

Als Hilfsprozeduren zur Erzielung eines klareren Quellcodes wurden die *IntegerEingabe* und die *IntegerAusgabe* einer einzelnen Integerzahl implementiert. Außerdem ist ein kleines Unterprogramm für die *Stringausgabe* enthalten.

Zur besseren Übersicht nun eine grafische Darstellung der Abhängigkeiten der einzelnen Unterprogramme (ich verwende auch gleich die Namen, wie sie später im Programm auftauchen):

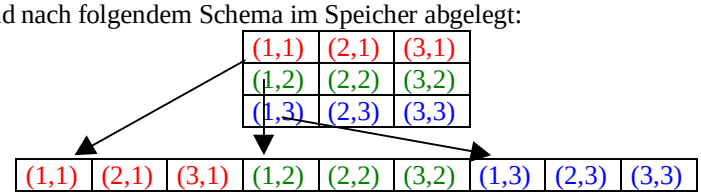


Die starke Strukturierung in 8 Unterprogramme führt dazu, dass die einzelnen Prozeduren sehr kurz und damit übersichtlich sind. Gleichzeitig erleichtert dies die spätere Konvertierung in eine Matrizenmultiplikation mit Gleitkommazahlen.

Hauptspeicherzugriff

Es müssen keine Variablen (von den Matrizen abgesehen) im Hauptspeicher gehalten werden, so sind alle, die im Datensegment auftauchen, nur Zeichenketten, die der Erläuterung von Ein- bzw. Ausgaben dienen.

Die Matrizen selber sind nach folgendem Schema im Speicher abgelegt:



Aus der rechteckigen, zweidimensionalen Matrix wird ein eindimensional, linear angeordnetes Array. Jedes Element umfaßt dabei 4 Bytes, da ich Integer- bzw. Float-Werte (dazu später mehr) speichere.
Die Adresse einer einzelnen Speicherzelle errechnet sich dann als:

$$adressoffset = 4 \cdot ((Zeile - 1) \cdot AnzahlSpalten + Spalte - 1)$$

Da meine Zählung bei 1 statt bei 0 beginnt, müssen von der Zeilen- und der Spaltennummer jeweils 1 subtrahiert werden. Dieser Offset wird schlußendlich zu der Startadresse der Matrix addiert.

Die Notation folgt stets dem Schema: Matrix(Spalte,Zeile).

Detaillierte Erklärung der Unterprogramme

Aufgrund der Kürze und programmtechnischen Trivialität von *Integereingabe*, *Integerausgabe* und *Stringausgabe* gehe ich auf sie nicht näher ein. Bezüglich *Zahleingabe* und *Zahlausgabe* verweise ich auf den Abschnitt ‚*Konvertierung in Gleitkommamatrizen*' weiter hinten im Text, für Integerzahlen ist ihre Funktion identisch mit ihren Namensvettern, die mit *Integer* beginnen.

Zu jeder Prozedur gebe ich die Parameter, mit denen sie mit der Umwelt kommuniziert (dabei seien übergebene mit →, übergebende mit ← gekennzeichnet), und die Veränderungen globaler Variablen an. Ich erwähne nicht Programmteile, die lediglich der optischen Abrundung des Programmes dienen und somit für die Funktion nicht-essentiell sind.

Zur Parameterübergabe werden die Register `$a0` bis `$a2` verwendet. Reichen diese nicht aus, so benutze ich die globalen Register `$s0` bis `$s4`, allerdings werden sie dabei nicht manipuliert. Die Rückgabewerte sind in der Regel globale Werte, so dass ich sie in ihre Zielregister `$s0` bis `$s5` statt in `$v0`/`$v1` speichere.
Es werden die Register `$s0` und `$s1` dauerhaft zweckgebunden belegt, dabei gilt `$s0`=*m* und `$s1`=*n*.
Außerdem speichere ich die Adressen der Matrizen in `$s2` (=*A*), `$s3` (=*B*) und `$s4` (=*C*).
Für Schleifen kommen die lokal frei verwendbarer Register `$t0` bis `$t5` zum Zuge. `$t6` und `$t7` dienen in den Unterprogrammen zur Speicherung der Rücksprungadresse, wenn sie andere aufrufen, da dann die alleinige Verwendung von `$ra` nicht ausreicht.

Eingabe

Parameter:	→	keine
	←	Dimension der Matrizen: `$s0` = m
		`$s1` = n
		`$s2` – Zeiger auf Speicherbereich von A
		`$s3` – Zeiger auf Speicherbereich von B
Global:	- legt Speicherbereiche für A und B an	
	- belegt indirekt (über Unterprogramme) A und B mit eingegebenen Werten	

Zuerst werden `$s0` und `$s1` eingelesen, die die Dimension von A bzw. B repräsentieren. Dies geschieht jeweils unter Benutzung der Prozedur *IntegerEingabe*.
Anschließend werden diese beiden Werte als Parameter nach `$a0` und `$a1` kopiert und zusammen mit der Adresse von A, die in `$s2` steht, an *Matrizeneingabe* übergeben. Danach wird die gleiche Prozedur erneut aufgerufen, es ändert sich die Adresse der Matrix, sie ist nun die von B (= `$s3`), außerdem müssen `$a0` und `$a1` nun vertauscht werden.

Matrizeneingabe

Parameter:	→	`$a0` – Anzahl Zeilen
		`$a1` – Anzahl Spalten
		`$a2` – Speicheradresse, an der die Matrix liegt
	←	keine

Global: - legt an der gegebenen Speicheradresse die Eingabewerte ab

Über die lokal frei benutzbaren Register `$t0` und `$t1` verschachtele ich zwei Schleifen. Die äußere durchläuft mit `$t0` alle Zeilen (von 1 bis `$a0`), die innere mit `$t1` alle Spalten (von 1 bis `$a1`). In jedem Durchlauf der inneren Schleife wird ein Integerwert eingelesen, nachdem der Benutzer dazu aufgefordert wurde. Um die zu adressierende Speicherzelle nicht in jedem Durchlauf komplett berechnen zu müssen, läuft in `$t5` ein Zeiger, der pro Eingabe um 4 inkrementiert wird.

Matrizenmultiplikation

Parameter: → `$a0` – Anzahl Zeilen von A (=*m*)
`$a1` – Anzahl Spalten von A (=*n*)
`$s2` – Speicheradresse, an der die Matrix A liegt
`$s3` – Speicheradresse, an der die Matrix B liegt
← `$s4` – Speicheradresse, an der die Matrix C liegt

Global: - legt Speicherbereich für C an (`$s4` enthält dann einen Zeiger auf den Bereich)
- speichert in C das Ergebnis der Matrizenmultiplikation von A und B ab

Über die lokal frei benutzbaren Register `$t0`, `$t1` und `$t2` verschachtele ich drei Schleifen. Die äußere durchläuft mit `$t0` alle Zeilen (von 1 bis `$a0`), die mittlere mit `$t1` alle Spalten (ebenfalls von 1 bis `$a0`) von C. Diese mittlere Schleife produziert als Ergebnis den neuen Wert von C an der Stelle (`$t0`, `$t1`). In jedem Durchlauf der inneren Schleife mit `$t2` wird schließlich

$$c_{ij} = a_i \cdot b_j = \sum_{k=1}^{n} a_{ik} b_{kj}$$

umgesetzt, indem jeweils die Produkte $a_{jk}b_{kj}$, d.h. A(`$t0`, `$t2`) · B(`$t2`, `$t1`) gebildet werden und ihre Summe in `$t3` gespeichert wird. Nach Beendigung der Schleife wird `$t3` in C(`$t0`, `$t1`) gespeichert.

Ausgabe

Parameter: → `$a0` = *m*
`$a2` - Adresse von C
← keine

Global: -

Zuerst wird `$a0` ausgelesen, da C die Dimension (*m*,*m*) hat. Dieser Wert wird als Parameter zusammen mit der Adresse von C an *Matrizenausgabe* übergeben.

Matrizenausgabe

Parameter: → `$a0` – Anzahl Zeilen
`$a1` – Anzahl Spalten
`$a2` – Speicheradresse, an der die Matrix liegt
← keine

Global: -

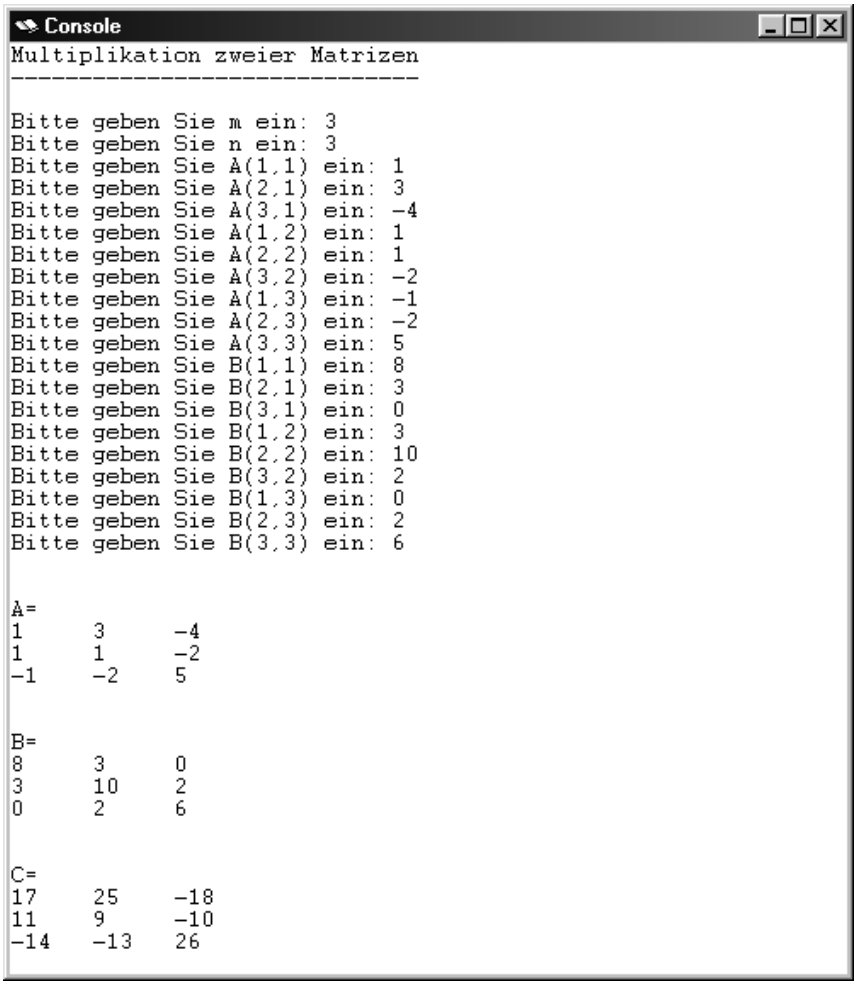
Über die lokal frei benutzbaren Register `$t0` und `$t1` verschachtele ich zwei Schleifen. Die äußere durchläuft mit `$t0` alle Zeilen (von 1 bis `$a0`), die innere mit `$t1` alle Spalten (von 1 bis `$a1`). In jedem Durchlauf der inneren Schleife wird ein Integerwert zusammen mit einer optischen Formatierung ausgegeben.
Man könnte jetzt einwerfen, dass nur C ausgegeben wird, wobei bekannt ist, dass dort die Anzahl Zeilen mit der Anzahl der Spalten identisch ist. Das ist zwar korrekt, aber meine verallgemeinerte Fassung kann beliebige Matrizen ausgeben, was besonders bei der Programmentwicklung hilfreich war.

Beispielrechnung

Für die gegebene Beispielaufgabe ergibt sich:

$$\begin{pmatrix} 1 & 3 & -4 \\ 1 & 1 & -2 \\ -1 & -2 & 5 \end{pmatrix} \bullet \begin{pmatrix} 8 & 3 & 0 \\ 3 & 10 & 2 \\ 0 & 2 & 6 \end{pmatrix} = \begin{pmatrix} 17 & 25 & -18 \\ 11 & 9 & -10 \\ -14 & -13 & 26 \end{pmatrix}$$

Genau dieses Ergebnis erzeugt auch mein Programm (Screenshot der Integer-Version, die Gleitkommafassung errechnet die gleiche Matrix):



Konvertierung in Gleitkommamatrizen

Bei der Konvertierung von einem Ganz- zu einem Gleitkommazahlprogramm ergab sich als erstes das Problem, welche Gleitkommadarstellung zu wählen sei. Um die Speicherberechnungen nicht ändern zu müssen, entschied ich mit für den Datentyp `float`, der, genau wie `integer`, 4 Bytes für die Darstellung benötigt.

Da ich schon von Anfang an die Gleitkommakonvertierung in den Programmentwurf einfließen ließ, war die eigentliche Änderung dann recht schnell durchzuführen. Die starke Unterteilung in viele Unterprogramme machte es möglich, dass nur *ZahlenEingabe*, *ZahlenAusgabe* und *ZahlenMultiplikationAddition* zu ändern waren. Ihre Schnittstellen mußten in keiner Art umgestaltet werden, daher erfolgt die Umstellung für den Rest des Programmes vollkommen transparent.

Um die Unterschiede deutlicher herauszustellen, folgen jetzt die Quelltexte der erwähnten Unterprogramme in farblicher Abstufung. Schwarze Zeilen stehen für Code, der in beiden Versionen auftaucht, roter ist in der Integer-Fassung aktiv, blauer in der Gleitkommaversion.

```
############################################################
# ZahlEingabe:
#   liest eine Zahl von der Konsole ein
#   IN:   -
#   OUT:  $v0 = eingelesene Zahl
ZahlEingabe:
       li      $v0,5                  # Systemaufruf-Code 5: Einlesen eines Integer in $v0
       li      $v0,6                  # Systemaufruf-Code 6: Einlesen eines Single-Floats in
$f0
       syscall                        # Aufruf durchführen, jetzt enthaelt $v0 die Eingabe
       mfc1    $v0,$f0                # Single-Float nach $v0 kopieren
       jr      $ra                    # springt zum Aufrufer zurueck

############################################################
# ZahlAusgabe:
#   gibt eine Zahl auf dem Bildschirm aus
#   IN:   $a0 = auszugebende Zahl
#   OUT:  -
ZahlAusgabe:
       li      $v0,1                  # Systemaufruf-Code 1: Ausgabe des Integer in $a0
       li      $v0,2                  # Systemaufruf-Code 2: Ausgabe des Single-Float in $f12
       mtc1    $a0,$f12               # Single-Float nach $f12 kopieren
       syscall                        # Aufruf durchführen
       jr      $ra                    # springt zum Aufrufer zurueck

############################################################
# ZahlenMultiplikationAddition:
#   multipliziert beide Faktoren und
#   addiert anschliessend das Produkt mit dem Summanden
#   IN:   $a0 = Faktor 1
#         $a1 = Faktor 2
#         $v0 = zu addierender Summand
#   OUT:  $v0 = Produkt
ZahlenMultiplikationAddition:
       mul     $a0,$a0,$a1            # Integermultiplikation
       add     $v0,$v0,$a0            # Interaddition

       mtc1    $a0,$f3                # Single-Float nach $f3 kopieren
       mtc1    $a1,$f4                # Single-Float nach $f4 kopieren
       mtc1    $v0,$f5                # Single-Float nach $f4 kopieren
       mul.s   $f3,$f3,$f4            # miteinander multiplizieren
       add.s   $f3,$f3,$f5            # anschliessend Summand addieren
       mfc1    $v0,$f3                # Ergebnis nach $v0 kopieren

       jr      $ra                    # springt zum Aufrufer zurueck
```

Die Gegenüberstellung ergibt:

Unterprogramm	Zeilen für Integer	Zeilen für Gleitkomma
ZahlEingabe	1	2
ZahlAusgabe	1	2
ZahlenMultiplikationAddition	2	6

Verglichen mit den ca. 400 Zeilen, die das gesamte Programm umfaßt, sind lediglich 4 speziell für Integer- bzw. 10 speziell für Gleitkommaoperationen verantwortlich.

Implementation des Programms als Unterprogramm

Laut Aufgabenstellung werden die Größe der Felder und ihre Adressen im Datenspeicher übergeben. Da diese in meinem Programm die ganze Zeit über in den Registern `$s0` bis `$s3` gespeichert werden, ist dafür lediglich das Kopieren von Registern notwendig.
Die Rückgabe ist bereits teilweise im Register `$s4` vorhanden, dort steht die Anfangsadresse der Ergebnismatrix. Ihre Endadresse errechnet sich aus ihrer Größe plus der Startadresse, d.h.

$$Endadresse = Startadresse + 4 \cdot m^2$$

Der Code ab dem Label *main* müßte dazu geändert werden in:

```
        move      $a0,$s0               # m als Parameter vorbereiten
        move      $a1,$s1               # n als Parameter vorbereiten
                        # Speicheradressen von A und B stehen bereits in $s2 bzw. $s3
        jal       MatrizenMultiplikation
                        # Matrizen miteinander multiplizieren, Ergebnis in $s5 (nur Zeiger !)
        move      $v0,$s4               # Startadresse in $v0 zurueckgeben
        li        $t0,4                 # Endadresse berechnen (nach obiger Formel)
        mul       $v1,$s0,$s0
        mul       $v1,$v1,$t0
        add       $v1,$v1,$v0           # und in $v1 zurueckgeben
```

Als Konvention vereinbare ich dabei, dass in `$v0` die Startadresse und in `$v1` die Endadresse der Ergebnismatrix zurückgeliefert wird.
Die Prozeduren für die Matrizenein- und ausgabe sind dann überflüssig und können entfallen.

Zusätzlich ist nun noch eine Sicherung der verwendeten Register auf dem Stack notwendig. Was durch geschickte Verwendung der Register bislang unnötig war, ist nun unabdingbar, da das Unterprogramm nicht wissen kann, welche Register im Hauptprogramm wofür verwendet werden. Insbesondere ist dabei auf die Register `$s0` bis `$s7` und `$ra` zu achten. Die Sicherung eines Registers könnte beispielsweise auf folgende Art und Weise erfolgen:

```
         sw      $s0,0($sp)
```

Dabei steht `$s0` nur stellvertretend für die anderen Register. Ebenso müsste der Offset 0 vor `($sp)` angepasst werden, wobei zu beachten ist, dass ein Register 4 Bytes umfasst.

Der Quellcode

Im folgenden der komplette Quellcode. Vereinzelt kommt es zu Zeilenumbrüchen innerhalb der Kommentare durch die verwendete Textverarbeitung, die aber im originalen Code nicht vorhanden sind.

```
############################################################
#
#   Hausarbeit zur MIPS-Programmierung
#
#   Bearbeiter: Stephan Brumme, Matrikelnr. 702544
#               techinf@stephan-brumme.de
#
#   letzte Aenderung: ??.Juli 2000
#
############################################################

        .data

# Allgemeine Begruessung
Startmeldung:  .asciiz "Multiplikation zweier Matrizen\n------------------------------\n\n"

# Eingabeaufforderungen, zerlegt, da Zahlen in die Ausgabe eingefuegt werden
Eingabe_m:     .asciiz "Bitte geben Sie m ein: "
Eingabe_n:     .asciiz "Bitte geben Sie n ein: "
Eingabe_Matr1: .asciiz "Bitte geben Sie "
Eingabe_Matr2: .asciiz "A("
Eingabe_Matr3: .asciiz ","
Eingabe_Matr4: .asciiz ") ein: "

# Ausgabemeldungen
Ausgabe_MatrA:  .asciiz "\n\nA=\n"
Ausgabe_MatrB:  .asciiz "\n\nB=\n"
Ausgabe_MatrC:  .asciiz "\n\nC=\n"
Ausgabe_Matr1:  .asciiz "\t"
Ausgabe_Matr2:  .asciiz "\n"



        .text

############################################################
```

```
# Hauptprogramm

main:
       la        $a0,Startmeldung      # Allgemeine Begruessung
       jal       StringAusgabe         # und ausgeben

       jal       Eingabe               # 2 Matrizen von der Konsole einlesen

       move      $a0,$s0               # m als Parameter vorbereiten
       move      $a1,$s1               # n als Parameter vorbereiten
                                       # Speicheradressen von A und B stehen bereits in $s2
bzw. $s3
       jal       MatrizenMultiplikation
                                       # Matrizen miteinander multiplizieren, Ergebnis in $s5
(nur Zeiger !)

       jal       Ausgabe               # A, B und C auf der Konsole ausgeben

       li        $v0,10                # Programm beenden
       syscall



############################################################
# Eingabe:
#   liest zwei komplette Matrizen von der Konsole ein
#   IN:   -
#   OUT:  $s0 = m
#         $s1 = n
#         $s2 = Zeiger auf Speicherbereich von A
#         $s3 = Zeiger auf Speicherbereich von B
Eingabe:
       move    $t7,$ra                 # Ruecksprungadresse sichern

       la      $a0,Eingabe_m           # lies nach Aufforderung m in $s0 ein
       jal     StringAusgabe
       jal     IntegerEingabe
       move    $s0,$v0                 # in $s0 speichern, wird dort nie mehr veraendert

       la      $a0,Eingabe_n           # lies nach Aufforderung n in $s1 ein
       jal     StringAusgabe
       jal     IntegerEingabe
       move    $s1,$v0                 # in $s1 speichern, wird dort nie mehr veraendert

       mul     $t0,$s0,$s1             # ermittle Speichergroesse von A und B
       li      $t1,4                   # pro Element sind 4 Bytes notwendig
       mul     $a0,$t0,$t1             # korrekte Groesse in $a0

       li      $v0,9                   # Speicherblock fuer A anfordern
       syscall
       move    $s2,$v0                 # Zeiger auf A in $s2 speichern

       li      $v0,9                   # Speicherblock fuer B anfordern, Groesse ist immer
noch in $a0
       syscall
       move    $s3,$v0                 # Zeiger auf B in $s3 speichern

       # Matrix A
       move    $a0,$s0                 # m Zeilen
       move    $a1,$s1                 # n Spalten
       move    $a2,$s2                 # Speicherbereich von A
       jal     MatrizenEingabe         # Matrixwerte einlesen

       li      $t0,66                  # ASCII-Code von B
       sb      $t0,Eingabe_Matr2       # direkt in die Zeichenkette patchen

       # Matrix B
       move    $a0,$s1                 # n Zeilen
       move    $a1,$s0                 # m Spalten
       move    $a2,$s3                 # Speicherbereich von B
       jal     MatrizenEingabe         # Matrixwerte einlesen

       jr $t7                          # zum Aufrufer zuruecksprungen
```

```
############################################################
# MatrizenEingabe:
#   liest eine Matrix von der Konsole ein
#   IN:   $a0 = Anzahl Zeilen
#         $a1 = Anzahl Spalten
#         $a2 = Zeiger auf den zu beschreibenden Speicherbereich
#   OUT:  -  (Veraenderung direkt im Hauptspeicher)
MatrizenEingabe:
       move    $t6,$ra                 # Ruecksprungadresse sichern

       move    $t2,$a0                 # um $a0 nicht durch Systemaufrufe zu verlieren,
speichere ich es in $t2
       move    $t3,$a1                 # zur besseren Uebersichtlichkeit speichere ich auch
$a1 in $t3

       move    $t5,$a2                 # Zeiger auf Speicherbereich laden

       li      $t0,1                   # aeussere Schleife, durchlaeuft alle Zeilen 1 bis $a0
(kopiert nach $t2)
ME_outer_loop:
       li      $t1,1                   # innere Schleife, durchlaeuft alle Spallen 1 bis $a1
(kopiert nach $t3)
ME_inner_loop:
       la      $a0,Eingabe_Matr1       # Eingabeaufforderung
       jal     StringAusgabe           # allgemeines Format: Matrixname(Spalte,Zeile)
       la      $a0,Eingabe_Matr2       # Matrixname ist A bzw. B
       jal     StringAusgabe
       move    $a0,$t1
       jal     IntegerAusgabe
       la      $a0,Eingabe_Matr3
       jal     StringAusgabe
       move    $a0,$t0
       jal     IntegerAusgabe
       la      $a0,Eingabe_Matr4
       jal     StringAusgabe           # Eingabeaufforderung komplett

       jal     ZahlEingabe             # Zahl einlesen
       sw      $v0,($t5)               # im Speicher ablegen
       addi    $t5,$t5,4               # Zeiger auf die naechste Speicherzelle weiterbewegen

       addi    $t1,$t1,1               # naechstes Element dieser Zeile
       ble     $t1,$t3,ME_inner_loop   # von 1 bis $t2 (urspruenglich $a1)

       addi    $t0,$t0,1               # naechste Spalte
       ble     $t0,$t2,ME_outer_loop   # von 1 bis $t2 (urspruenglich $a0)

       jr      $t6                     # zum Aufrufer zurueckspringen


############################################################
# MatrizenMultiplikation:
#   multipliziert zwei Matrizen der Form ($a0,$a1) und ($a1,$a0)
#   das Ergebnis ist eine Matrix der Form ($a0,$a0)
#   IN:   $a0 = Anzahl Zeilen
#         $a1 = Anzahl Spalten
#         $s2 = Zeiger auf Faktor Matrix A
#         $s3 = Zeiger auf Faktor Matrix B
#   OUT:  $s4 = Zeiger auf Produkt Matrix C
MatrizenMultiplikation:
       move    $t7,$ra                 # Ruecksprungadresse sichern

       move    $a2,$a0                 # um $a0 nicht durch UP-Aufrufe zu verlieren, speichere
ich es in $a2
       move    $a3,$a1                 # um $a1 nicht durch UP-Aufrufe zu verlieren, speichere
ich es in $a3

       mul     $t0,$a0,$a0             # ermittle Speichergroesse von C
       li      $t1,4                   # pro Element sind 4 Bytes notwendig
       mul     $a0,$t0,$t1             # reale Groesse nun in $a0

       li      $v0,9                   # Speicherblock fuer C anfordern
       syscall
```

```
        move    $s4,$v0                 # Zeiger auf C in $s4 speichern

        move    $t5,$s4                 # Zeiger auf C laden

        li      $t0,1                   # aeussere Schleife, Zeilen von 1 bis $a0 (nun $a2)

MM_outer_loop:
        li      $t1,1                   # mittlere Schleife, Spalten von 1 bis $a1 (nun $a3)

MM_middle_loop:
        move    $t3,$t0                 # Zeiger auf A(1,$t0) laden
        addi    $t3,$t3,-1              # $t3=Zeile-1
        mul     $t3,$t3,$a3             # $t3=(Zeile-1)*AnzahlSpalten
        li      $t4,4
        mul     $t3,$t3,$t4             # $t3=(Zeile-1)*AnzahlSpalten*4
        add     $t3,$t3,$s2             # $t3=(Zeile-1)*AnzahlSpalten*4+BasisadresseMatrixA

        li      $t4,4                   # Zeiger auf B($t1,1) laden
        mul     $t4,$t4,$t1             # $t4=Spalte*4
        addi    $t4,$t4,-4              # da bereits mit 4 multipliziert wurde, nun 4 statt 1
subtrahieren
                                        # d.h. $t4=(Spalte-1)*4
        add     $t4,$t4,$s3             # $t4=(Spalte-1)*4+BasisadresseMatrixA

        li      $t2,1                   # innere Schleife, alle Terme aufsummieren

        li      $v0,0                   # Summe mit 0 initialisieren
MM_inner_loop:
        lw      $a0,($t3)               # A($t0,$t2) laden
        lw      $a1,($t4)               # B($t2,$t1) laden
        jal     ZahlenMultiplikationAddition
                                        # miteinander multiplizieren und Produkt zu $v0
addieren

        addi    $t3,$t3,4               # naechstes Element von A (da jeweils 4 Bytes)

        add     $t4,$t4,$a3             # naechstes Element von B (da jeweils 4 Bytes)
        add     $t4,$t4,$a3             # um Register zu sparen, addiere ich 4mal $a3
        add     $t4,$t4,$a3             # d.h. die Anzahl der Spalten
        add     $t4,$t4,$a3

        addi    $t2,$t2,1               # naechsten Term dazuaddieren
        ble     $t2,$a3,MM_inner_loop   # von 1 bis $a2 => Anzahl Spalten von A, Zeilen von B


        sw      $v0,($t5)               # Element von C ist berechnet, also in der Matrix
speichern
        addi    $t5,$t5,4               # naechstes Element von C (da jeweils 4 Bytes)

        addi    $t1,$t1,1               # naechstes Element dieser Zeile
        ble     $t1,$a2,MM_middle_loop  # von 1 bis $a2 (Anzahl Spalten)

        addi    $t0,$t0,1               # naechste Spalte
        ble     $t0,$a2,MM_outer_loop   # von 1 bis $a2 (Anzahl Zeilen, identisch mit Anzahl
Spalten)

        jr      $t7                     # zum Aufrufer zurueckspringen


############################################################
# Ausgabe:
#   gibt eine komplette Matrix auf der Konsole aus
#   IN:   $s0 = m (Anzahl Zeilen und Spalten ist bei C identisch)
#         $s1 = n (Anzahl Zeilen und Spalten ist bei C identisch)
#         $s2 = Zeiger auf den Speicherbereich, an dem die Matrix A liegt
#         $s3 = Zeiger auf den Speicherbereich, an dem die Matrix B liegt
#         $s4 = Zeiger auf den Speicherbereich, an dem die Matrix C liegt
#   OUT:  -  (Konsolenausgabe)
Ausgabe:
        move      $t7,$ra               # Ruecksprungadresse sichern

        la        $a0,Ausgabe_MatrA
        jal       StringAusgabe
        move      $a0,$s0               # m als Parameter vorbereiten
```

```
       move      $a1,$s1              # n als Parameter vorbereiten
       move      $a2,$s2              # Speicheradresse von A uebergeben
       jal       MatrizenAusgabe      # Ergebnis auf der Konsole ausgeben

       la        $a0,Ausgabe_MatrB
       jal       StringAusgabe
       move      $a0,$s1              # n als Parameter vorbereiten
       move      $a1,$s0              # m als Parameter vorbereiten
       move      $a2,$s3              # Speicheradresse von B uebergeben
       jal       MatrizenAusgabe      # Ergebnis auf der Konsole ausgeben

       la        $a0,Ausgabe_MatrC
       jal       StringAusgabe
       move      $a0,$s0              # m als Parameter vorbereiten
       move      $a1,$s0              # m als Parameter vorbereiten
       move      $a2,$s4              # Speicheradresse von C uebergeben
       jal       MatrizenAusgabe      # Ergebnis auf der Konsole ausgeben

       jr $t7                         # zum Aufrufer zurueckspringen


############################################################
# MatrizenAusgabe:
#   gibt eine Matrix auf der Konsole aus
#   IN:   $a0 = Anzahl Zeilen
#         $a1 = Anzahl Spalten
#         $a2 = Zeiger auf den auszugebenden Speicherbereich
#   OUT:  -  (Konsolenausgabe)
MatrizenAusgabe:
       move    $t6,$ra                # Ruecksprungadresse sichern

       move    $t2,$a0                # um $a0 nicht durch Systemaufrufe zu verlieren,
speichere ich es in $t2
       move    $t3,$a1                # um $a1 nicht durch Systemaufrufe zu verlieren,
speichere ich es in $t3

       move    $t5,$a2                # Zeiger auf Speicherbereich laden

       li      $t0,1                  # aeussere Schleife, Zeilen 1 bis $a0 durchlaufen
MA_outer_loop:
       li      $t1,1                  # innere Schleife, Spalten 1 bis $a1 durchlaufen
MA_inner_loop:
       lw      $a0,($t5)              # Zahl aus Speicher auslesen
       jal     ZahlAusgabe            # und auf der Konsole ausgeben
       la      $a0,Ausgabe_Matr1      # Tabulator ausgeben, um Zahlen besser lesen zu koennen
       jal     StringAusgabe
       addi    $t5,$t5,4              # Zeiger auf die naechste Speicherzelle weiterbewegen

       addi    $t1,$t1,1              # naechstes Element dieser Zeile
       ble     $t1,$t3,MA_inner_loop  # von 1 bis $t3=$a1

       la      $a0,Ausgabe_Matr2      # Zeilenumbruch ausgeben
       jal     StringAusgabe

       addi    $t0,$t0,1              # naechste Spalte
       ble     $t0,$t2,MA_outer_loop  # von 1 bis $t2=$a0

       jr      $t6                    # zum Aufrufer zurueckspringen



############################################################
############################################################
##   Hilfsunterprogramme


############################################################
# ZahlenMultiplikationAddition:
#   multipliziert beide Faktoren und
#   addiert anschliessend das Produkt mit dem Summanden
#   IN:   $a0 = Faktor 1
#         $a1 = Faktor 2
```

```
#         $v0 = zu addierender Summand
#   OUT:  $v0 = Produkt
ZahlenMultiplikationAddition:
       mul     $a0,$a0,$a1             # Integermultiplikation
       add     $v0,$v0,$a0             # Interaddition

#       mtc1    $a0,$f3                 # Single-Float nach $f3 kopieren
#       mtc1    $a1,$f4                 # Single-Float nach $f4 kopieren
#       mtc1    $v0,$f5                 # Single-Float nach $f4 kopieren
#       mul.s   $f3,$f3,$f4             # miteinander multiplizieren
#       add.s   $f3,$f3,$f5             # anschliessend Summand addieren
#       mfc1    $v0,$f3                 # Ergebnis nach $v0 kopieren

       jr      $ra                     # springt zum Aufrufer zurueck



############################################################
# ZahlEingabe:
#   liest eine Zahl von der Konsole ein
#   IN:   -
#   OUT:  $v0 = eingelesene Zahl
ZahlEingabe:
       li      $v0,5                   # Systemaufruf-Code 5: Einlesen eines Integer in $v0
#       li      $v0,6                   # Systemaufruf-Code 6: Einlesen eines Single-Floats in
$f0
       syscall                         # Aufruf durchführen, jetzt enthaelt $v0 die Eingabe
#       mfc1    $v0,$f0                 # Single-Float nach $v0 kopieren
       jr      $ra                     # springt zum Aufrufer zurueck



############################################################
# ZahlAusgabe:
#   gibt eine Zahl auf dem Bildschirm aus
#   IN:   $a0 = auszugebende Zahl
#   OUT:  -
ZahlAusgabe:
       li      $v0,1                   # Systemaufruf-Code 1: Ausgabe des Integer in $a0
#       li      $v0,2                   # Systemaufruf-Code 2: Ausgabe des Single-Float in
$f12
#       mtc1    $a0,$f12                # Single-Float nach $f12 kopieren
       syscall                         # Aufruf durchführen
       jr      $ra                     # springt zum Aufrufer zurueck



############################################################
# IntegerEingabe:
#   liest eine Integerzahl von der Konsole ein
#   IN:   -
#   OUT:  $v0 = eingelesene Zahl
IntegerEingabe:
       li      $v0,5                   # Systemaufruf-Code 5: Einlesen eines Integer in $v0
       syscall                         # Aufruf durchführen, jetzt enthaelt $v0 die Eingabe
       jr      $ra                     # springt zum Aufrufer zurueck



############################################################
# IntegerAusgabe:
#   gibt eine Integerzahl auf dem Bildschirm aus
#   IN:   $a0 = auszugebende Zahl
#   OUT:  -
IntegerAusgabe:
    li      $v0,1                      # Systemaufruf-Code 1: Ausgabe des Integer in $a0
    syscall                            # Aufruf durchführen
    jr      $ra                        # springt zum Aufrufer zurueck



############################################################
# StringAusgabe:
#   gibt eine Zeichenkette auf dem Bildschirm aus
```

```
#   IN:   $a0 = Adresse der auszugebenden Zeichenkette
#   OUT:  -
StringAusgabe:
    li      $v0,4                   # Systemaufruf-Code 4: Ausgabe der Zeichenkette an der
Adresse in $a0
    syscall                         # Aufruf durchführen
    jr      $ra                     # springt zum Aufrufer zurueck
```