10.4. Matrices

unsigned char M[2][3];
// byte matrix with 2 rows and 3 columns

row I

column J

0 1 2

M[0,0] M[0,1] M[0,2]
M[1,0] M[1,1] M[1,2]

Figure 10.3. A byte matrix with 2 rows and 3 columns.

With row major allocation,
the elements of each row are stored together

I be the row index,
J be the column index,

n be the number of bytes in each row

Base is the base address of the byte matrix
then the address of the element at M[I,J] is

Base+n*I+J

The row index (0 or 1) is passed in Register A.
The column index (0, 1, or 2) is passed in Register B.
The base address of the matrix is passed in Register X.
The subroutine returns the value in Register A.

READ pshb        Allocate J and initialize
      ldab #3         number of columns
mul 3*I
addb 1,s+ 3*I+J
ldaa B,X read value at M[I,J]
rts

Program 10.8. Assembly function to access a two by three row-major matrix.

Example read M[1,2]

<table>
<thead>
<tr>
<th>row I</th>
<th>column J</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$0910 $0911 $0912 $0913 $0914 $0915</td>
</tr>
<tr>
<td>1</td>
<td>M[0,0] M[0,1] M[0,2] $0910 $0911 $0912</td>
</tr>
<tr>
<td></td>
<td>M[1,0] M[1,1] M[1,2] $0913 $0914 $0915</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Desk check is a paper and pencil test of the program

ldaa #1 A=I=1
ldab #2 B=J=2
ldx #M X=$0910
bsr Read

READ pshb Save J
ldab #3 B=3 number of columns
mul B=3 3*I
addb 1,s+ B=5 3*I+J
ldaa B,X A=[$0915] EA=X+B=$0915
rts

With column major allocation, the elements of each column are stored together.
I be the row index,
J be the column index,
m be the number of bytes in each column , and
Base is the base address of the byte matrix, then the address of the element at \( M[I,J] \) is 
\[ Base + m \times J + I \]

The row index (0 or 1) is passed in Register A. The column index (0, 1, or 2) is passed in Register B. The base address of the matrix is passed in Register X. The subroutine returns the value in Register A.

```
READ  aslb       Reg B = 2*J
       abx        Reg X = base + 2*J
       ldaa A,X   read value at M[I,J]
       rts
```

Program 10.9. Assembly function to access a two by three column-major matrix.

Example read \( M[1,1] \)

<table>
<thead>
<tr>
<th>row I</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>M[0,0]</td>
<td>M[0,1]</td>
<td>M[0,2]</td>
</tr>
<tr>
<td>1</td>
<td>M[1,0]</td>
<td>M[1,1]</td>
<td>M[1,2]</td>
</tr>
</tbody>
</table>

\[ \text{column J} \]
0 1 2
\[ \text{column 0} \]
\[ \text{column 1} \]
\[ \text{column 2} \]

Desk check is a paper and pencil test of the program

```
ldaa #1       A=I=1
ldab #1       B=J=1
ldx   #M      X=$0910
bsr  Read
READ  aslb     B=2 = 2*J
           abx      X=$0912 = base + 2*J
           ldaa A,X A=$[0913], EA=A+X=$0913
```
rts

With a word row-major matrix, each element requires two bytes of storage.
I be the row index,
J be the column index,
n be the number of words in each row, and
Base is the base address of the word matrix, then the address of the element at \( M[I,J] \) is
\[
\text{Base}+2*(n*I+J)
\]

An assembly language program that accesses elements from a word matrix defined in row-major format

* The matrix is m rows by n columns
* ROW MAJOR Address = Base+2*(n*I+J)
* Zero-origin indexing Each element 2 bytes
n equ 10 the number of columns
*Input: Reg A is the row index (I)
* Reg B is the column index (J)
* Reg X is Base, points to first
*Output: Reg X points to the (I,J) element
* Local variables
Base set 0 address of first
J set 2 column index
Access pshb allocate and init J
pshx allocate and init Base
tsx stack frame pointer
ldab #n B is number of columns
mul n*I
addb J, X  n*I+J
adca #0
lsld 2*(n*I+J)
addw Base, X  D = Base + 2*(n*I+J)
xgdx X = Base + 2*(n*I+J)
leas 3, s
rts

Program 10.10. Assembly function to access a word matrix defined in row-major format.

Bit arrays can be used to store pixel values for graphics. Placing a 0 into a bit location will display a blank. Placing a 1 into a bit location will display that pixel. In assembly, we define the following in global RAM,

Video rmb 48

Figure 10.4. A bit-matrix with 12 rows and 32 columns.
Let $I$ be the row index, where $I$ ranges from 0 to 11. There are 4 bytes required in each row. Therefore, the starting address of row $I$ is

\[ \text{Video} + 4*I \]

Let $J$ be the column index, where $J$ ranges from 0 to 31. The column index specifies which byte, as well as which bit within that byte. The address of the byte containing the information at $(I, J)$ is

\[ \text{Video} + 4*I + J>>3 \]

where the divide by 8 is integer math without rounding. Notice that if $J$ is less than or equal to 31, then $J$ divided by 8 will be less than or equal to 3. Let $K$ be the bottom three bits of $J$.

\[ K = J\&0x07; \]

A mask will specify the bit location within the byte.

\[ \text{Masks fcb } \$80,\$40,\$20,\$10,\$08,\$04,\$02,\$01 \]

For example, if the bottom three bits are 010\(_2\) then we use the bit mask of $\$20$ to access the information

\[ \text{mask} = \text{Masks}[K]; \]

Program 10.12 takes the row and column index values and calculates the memory address and bit mask to access that bit in the Video matrix.

* ********** Access ******************
* Access the Video bit at (I,J)
*Input: A is the row index(I is 0 to 11)
* B is the column index (J is 0 to 31)
* Output: X points to the byte of interest
* A is the Mask to access that bit

**Access lsla**

- `lsla 4*I`
- `pshb` save a copy of J
- `lsrb`
- `lsrb B = J>>3
- `aba A = 4*I + J>>3
- `ldx #Video`
- `leax A,x X = Video + 4*I + J>>3`
- `pulb B = J again`
- `andb #$07 B = K (bottom 3 bits)`
- `ldy #Masks Y => Masks[0]`
- `ldaa B,Y A = mask = Masks[K]`
- `rts`

*Program 10.12. A helper function to access a bit-matrix.*

Functions to clear, set toggle bits in the Video matrix are shown in Program 10.13. A function that tests the current value within the matrix is shown in Program 10.14.

* Clear the Video bit at (I,J)
* Input: A is the row index (I is 0 to 11)
* B is the column index (J is 0 to 31)

**ClrBit bsr Access**

- `coma Not(mask) zero in bit`
- `anda 0,x Clear bit`
- `staa 0,x`
- `rts`
* Set the Video bit at (I,J)
* Input: A is the row index (I is 0 to 11)
* B is the column index (J is 0 to 31)

SetBit bsr Access

ora 0,x  Set bit
staa 0,x
rts

For example, say we wished to set this bit

```
$0A00  $SBBE4
$0A02  $2070
$0A04  $B8A8
$0A06  $2088
$0A08  $2F88
$0A0A  $2088
$0A0C  $8A08
$0A0E  $2088
$0A10  $8BBF
$0A12  $BE70
$0A14  $0000
$0A16  $0000
$0A18  $0000
$0A1A  $0000
$0A1C  $89CF
$0A1E  $20F0
$0A20  $8A28
$0A22  $A088
$0A24  $SA3F
$0A26  $2088
$0A28  $522A
$0A2A  $2088
$0A2C  $51C9
$0A2E  $3EE0
```

Desk check is a paper and pencil test of the program

```
ldaa #2       A=2
ldab #26      B=26 (00011010)
bsr  SetBit
setbit bsr  access
access lsla   A=4
lsla          A=8
pshb          save a copy of J
lsrb          B=13 (00001101)
lsrb          B=6  (00000110)
lsrb          B=3, (00000011) J>>3
aba           A=11 = 4*I+J>>3
```
ldx  #Video   X=$0A00
leax A,x      X=$0A0B =Video+4*I+J>>3
pulb          B=26 (00011010) = J again
andb #$07     B=2   K (bottom 3 bits)
ldy  #Masks   Y => Masks[0]
ldaa B,Y      A=$20 = Masks[K]
rts           [$0A0B]=$88
oraa 0,x      A=$A0
staa 0,x      [$0A0B]=$A8
rts