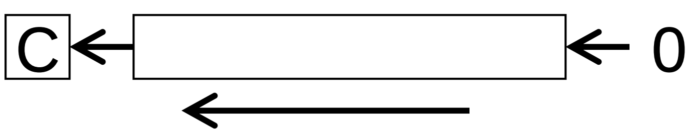
12 ARM Programming 3
CSC 230

Department of Computer Science
University of Victoria

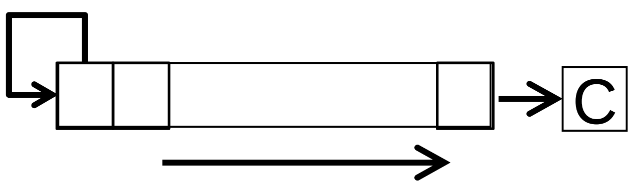
SHIFT OPERATIONS (conceptually)

Shift Left by n bits: logical or arithmetic shift - LSL



Multiplication by 2^n

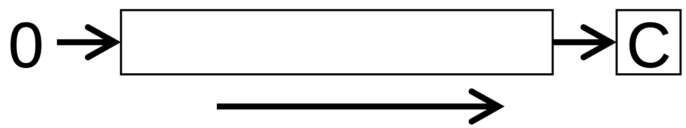
Arithmetic Shift Right by n bits - ASR



Signed division by 2^n

Logical Shift Right by n bits - LSR

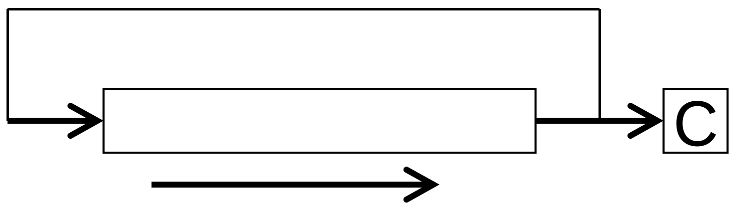
Unsigned division by 2^n



ROTATE INSTRUCTIONS: (conceptually)

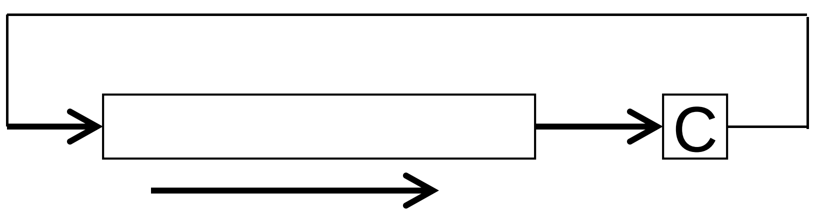
Rotate Right by n bits - ROR

32 bit rotate



Rotate Right extended by one bit - RRX

33 bit rotate, 33rd bit is carry



Shift and Rotate in ARM?

No explicit instructions

Shifts and rotate are incorporated into addressing modes

Examples:

```
MOV   R2,R2,LSL #4    @shift content of R2 left by 4 bits
                      @ that is, R2 = R2 * 16

ADD   R2,R2,R1,LSL #4          @R2 = R2 + R1 * 16

MOV   R2,R1,ASR #2             @R2 = R1 / 4

LDR   R2,[R1,R3,LSL #2]   @load from address
                          @calculated as R1 + R3 * 4
```

Where do the extra bits shifted out go?

```
MOV   R1,R2,ASR #1

          R2 =  00 00 12 34 in hex

   R2 =  0000 0000 0000 0000 0001 0010 0011 0100

   R1 =  0000 0000 0000 0000 0000 1001 0001 1010  0
```

LOST!

Where do the extra bits shifted out go?

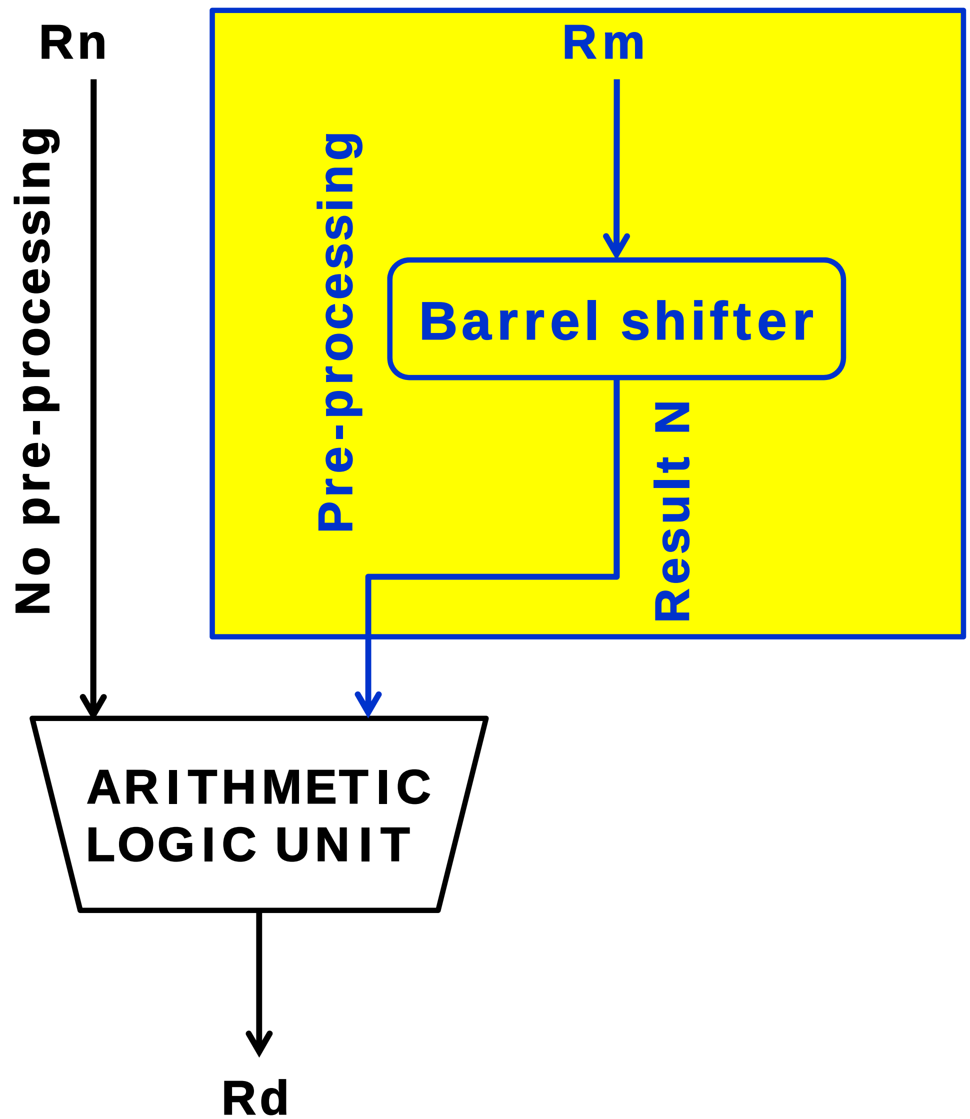
```
MOVS  R1,R2,ASR #1

              R2 =  00 00 12 34 in hex

   R2 =  0000 0000 0000 0000 0001 0010 0011 0100

   R1 =  0000 0000 0000 0000 0000 1001 0001 1010  0
```

CARRY bit in CPSR

If there is a shift of many positions, the last bit shifted is found in the Carry bit

Useful to test for parity (odd or even in rightmost bit) or count the number of non-zero bits ➔ use BCS or BCC

How are SHIFTS implemented in hardware for ARM?



Small Example: an ARM program for adding numbers

HVZ p. 119, fig. 3.8

```
@ ========== Data ==========
    .data      @ Begin the "data" segment, for variables
    .align     @ Next item begins at a word (aligned)
               @ address
sum:   .word 0
n:     .word 5

num1:  .word 3,-17,27,-12,322

uninit:   .skip 4
    .end
```

1 word initialized to 0

1 word initialized to 5

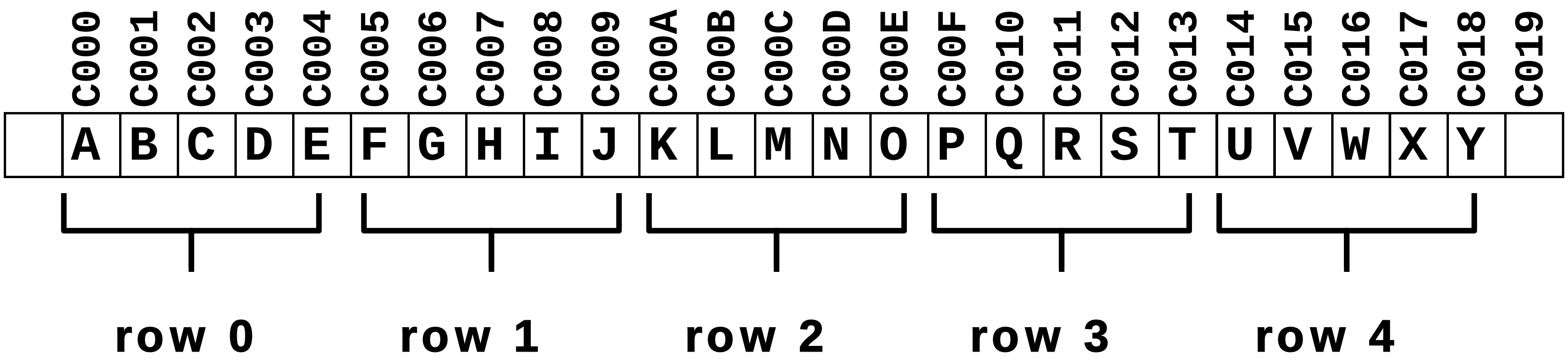
5 words initialized as shown (array)

```
@  Sample ARM to add a set of numbers
   .text       @ Begin the "text" (code) segment
   .global  _start   @ Export "_start" symbolic
                     @ address for linker
_start:
   ldr      r4,=n    @ Load address of var 'n'
   ldr      r1,[r4]  @ Load value of 'n'
   ldr      r2,=num1 @ Load address of num1
   mov      r0,#0    @ Initialize R0
loop: ldr   r3,[r2],#4  @ *** Note: post-indexed form
@ Here R2 should have been incremented by 4
   add      r0,r0,r3    @ update sum in r0
   subs  r1,r1,#1    @ r1=r1-1, plus condition codes
   bgt      loop     @ Loop again if R1 > 0
@ After loop:
   ldr      r4,=sum     @ Load address of var 'sum'
   str      r0,[r4]     @ Save value of R0 into sum
   swi      0x11
```

Study by yourself

Example 1: Printing a 2-D array of characters

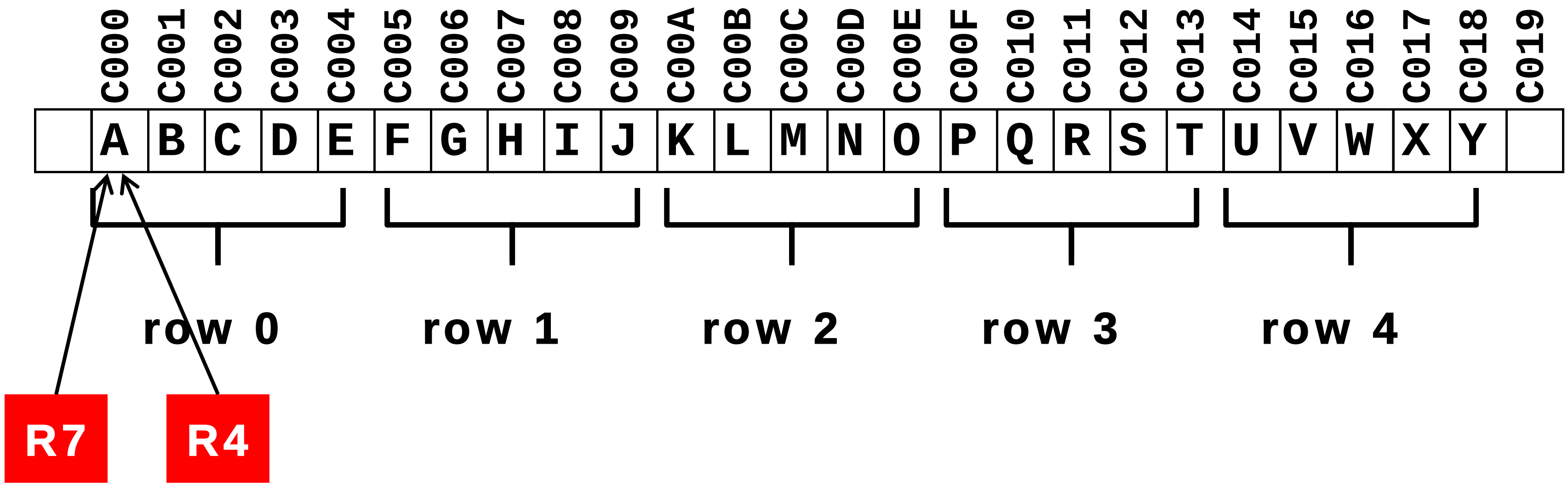
```
   .data
Rsize:  .word   5
Csize:  .word   5
MyArray2: .skip  25  @25 bytes for a 5x5 array of characters
```




```
FOR  ( rr=0; rr < Rsize; rr++ )
      FOR  ( cc=0; cc < Csize; cc++ )
          print MyArray2 [rr] [cc]
```

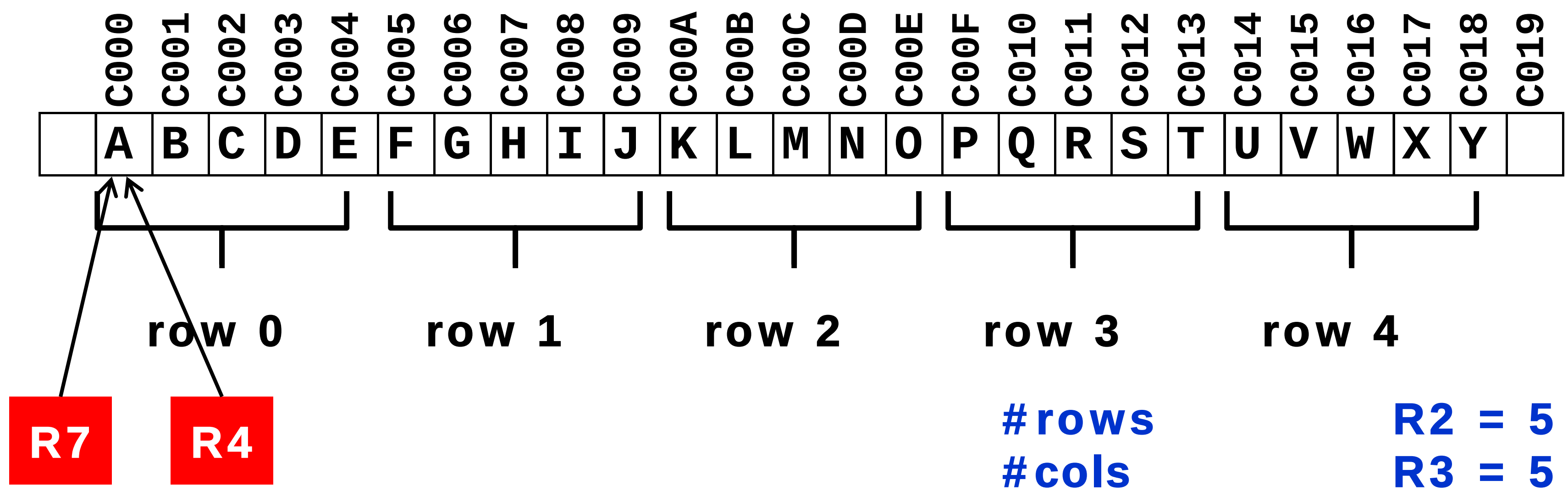
A	*B*	*C*	*D*	*E*
F	*G*	*H*	*I*	*J*
K	*L*	*M*	*N*	*O*
P	*Q*	*R*	*S*	*T*
U	*V*	*W*	*X*	*Y*

Example 1: Printing a 2-D array of characters




```
LDR    R7,=MyArray2   @ R7 has base address
LDR    R2,=Rsize
LDR    R2,[R2]        @ R2 has # rows, here =5
LDR    R3,=Csize
LDR    R3,[R3]        @ R3 has # columns, here =5

mov    r4,r7          @ R4 is also pointer to array base
```

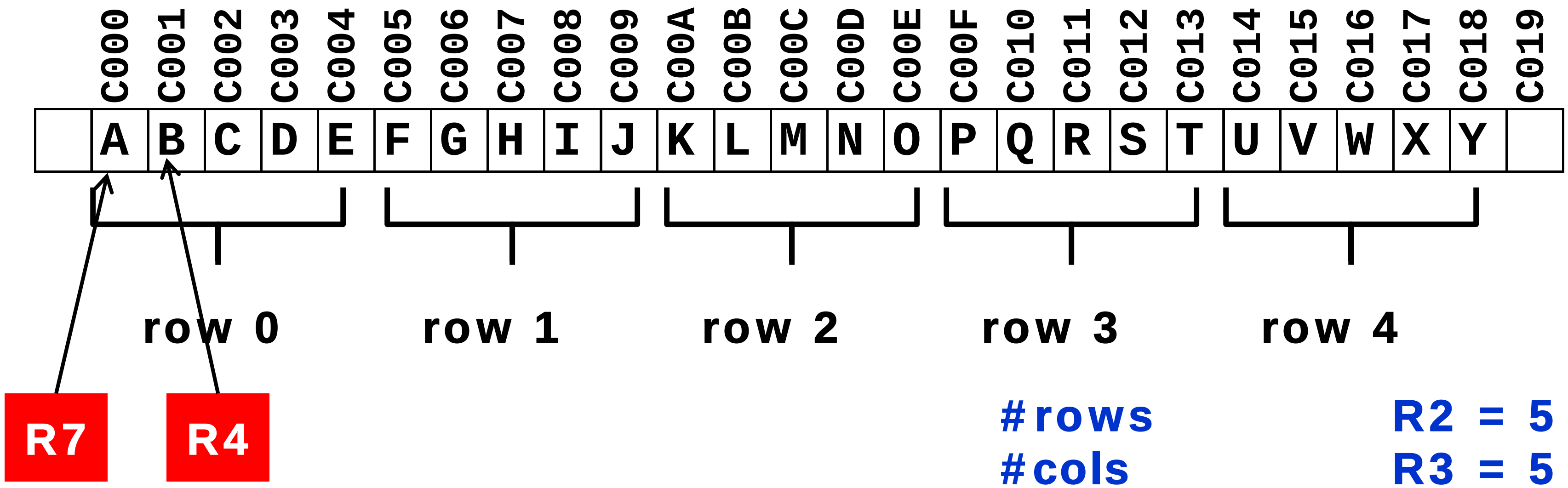
Example 1: Printing a 2-D array of characters



#rows R2 = 5
#cols R3 = 5
#cols counter R5 = 5

```
ROWLOOP:
        mov     r5,r3           @ set column counter
COLLOOP:
        ldrb    r0,[r4],#1      @ get char to be printed
        swi     SWI_PrChr       @ print it
        subs    r5,r5,#1        @ next element, same row?
        bne     COLLOOP
        ldr     r1, =EOL        @ end of line
        mov     R0,#Stdout      @ mode is Output view
        swi     SWI_PrStr
        subs    r2,r2,#1        @ next row?
        bne     ROWLOOP
```

Example 1: Printing a 2-D array of characters




```
ROWLOOP:
        mov     r5,r3          @ set column counter
COLLOOP:
        ldrb    r0,[r4],#1     @ get char to be printed
        swi     SWI_PrChr      @ print it
        subs    r5,r5,#1       @ next element, same row?
        bne     COLLOOP
        ldr     r1, =EOL       @ end of line
        mov     R0,#Stdout     @ mode is Output view
        swi     SWI_PrStr
        subs    r2,r2,#1       @ next row?
        bne     ROWLOOP
```

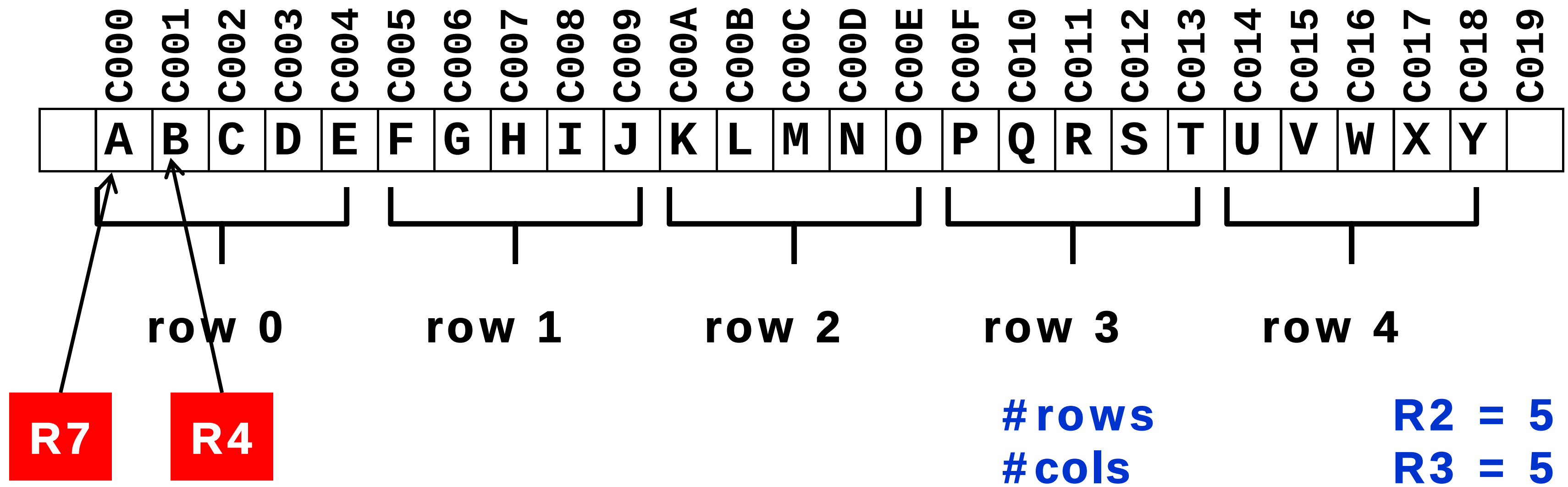
#rows **R2 = 5**
#cols **R3 = 5**
#cols counter **R5 = 5**

R0 = ‘A’

R4 = C001

Print ‘A’

Example 1: Printing a 2-D array of characters



#rows R2 = 5

#cols R3 = 5

#cols counter R5 = 4

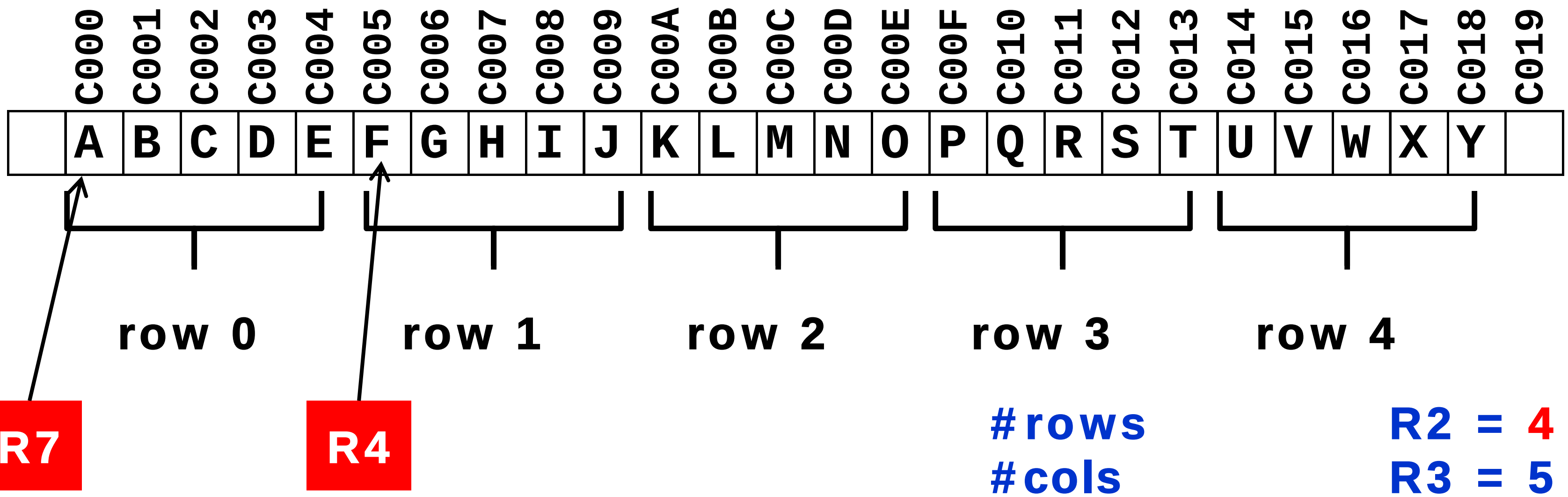
```
ROWLOOP:
        mov     r5,r3           @ set column counter
COLLOOP:
        ldrb    r0,[r4],#1      @ get char to be printed
        swi     SWI_PrChr       @ print it
        subs    r5,r5,#1        @ next element, same row?
        bne     COLLOOP
        ldr     r1, =EOL        @ end of line
        mov     R0,#Stdout      @ mode is Output view
        swi     SWI_PrStr
        subs    r2,r2,#1        @ next row?
        bne     ROWLOOP
```

R0 = ‘A’

R4 = C001

Loop until R5=0

Example 1: Printing a 2-D array of characters



#rows R2 = 4
#cols R3 = 5
#cols counter R5 = 0

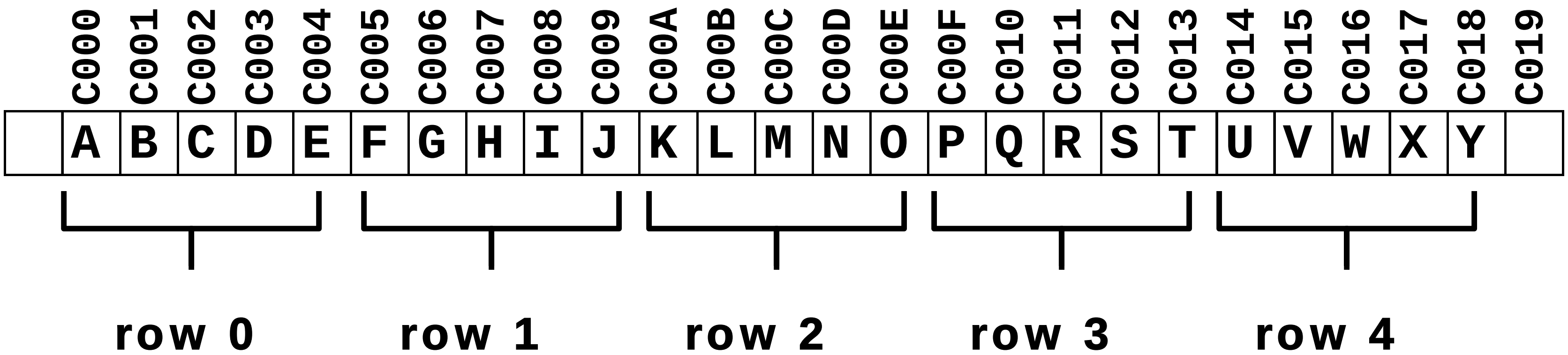
```
ROWLOOP:
        mov     r5,r3           @ set column counter
COLLOOP:
        ldrb    r0,[r4],#1      @ get char to be printed
        swi     SWI_PrChr       @ print it
        subs    r5,r5,#1        @ next element, same row?
        bne     COLLOOP
        ldr     r1, =EOL        @ end of line
        mov     R0,#Stdout      @ mode is Output view
        swi     SWI_PrStr
        subs    r2,r2,#1        @ next row?
        bne     ROWLOOP
```

R0 = ‘E’

R4 = C005

Next row

Example 1: Printing a 2-D array of characters



Continue tracing manually until you are convinced that you understand what is happening

```
ROWLOOP:
      mov    r5,r3          @ set column counter
COLLOOP:
      ldrb   r0,[r4],#1     @ get char to be printed
      swi    SWI_PrChr      @ print it
      subs   r5,r5,#1       @ next element, same row?
      bne    COLLOOP
      ldr    r1, =EOL       @ end of line
      mov    R0,#Stdout     @ mode is Output view
      swi    SWI_PrStr
      subs   r2,r2,#1       @ next row?
      bne    ROWLOOP
```

Example 2: Copy row i of Mat1 to row j of Mat 2

Given 2 matrices, i.e. 2 dimensional arrays of integers

C000	C004	C008	C00C	C010	C014	C018	C01C	C020	C024	C028	C02C	C030	C034	C038	C03C	C040	C044	C048	C04C	C050	C054
11	12	13	14	15	21	22	23	24	25	31	32	33	34	35	41	42	43	44	45		

R2

$$MAT1\ (4x5) = \begin{bmatrix} 11 & 12 & 13 & 14 & 15 \\ 21 & 22 & 23 & 24 & 25 \\ 31 & 32 & 33 & 34 & 35 \\ 41 & 42 & 43 & 44 & 45 \end{bmatrix}$$

stored as shown
With R2 = 0000C000

C124	C128	C12C	C130	C134	C138	C13C	C140	C144	C148	C14C	C150	C154	C158	C15C	C160	C164	C168	C16C	C170	C174	C178
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		

R3

$$MAT2\ (4x5) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

stored as shown
With R3 = 0000C124

Example 2: Copy row i of Mat1 to row j of Mat 2

Given 2 matrices, i.e. 2 dimensional arrays of integers
and given row i = 2 = R4 and row j = 3 = R5

C000	C004	C008	C00C	C010	C014	C018	C01C	C020	C024	C028	C02C	C030	C034	C038	C03C	C040	C044	C048	C04C	C050	C054
11	12	13	14	15	21	22	23	24	25	31	32	33	34	35	41	42	43	44	45		

$$MAT1\ (4x5) = \begin{bmatrix} 11 & 12 & 13 & 14 & 15 \\ 21 & 22 & 23 & 24 & 25 \\ 31 & 32 & 33 & 34 & 35 \\ 41 & 42 & 43 & 44 & 45 \end{bmatrix}$$

row i=2

C124	C128	C12C	C130	C134	C138	C13C	C140	C144	C148	C14C	C150	C154	C158	C15C	C160	C164	C168	C16C	C170	C174	C178
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		

$$MAT2\ (4x5) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

row j=3

```
@ *** CopyRow(Mat1,Mat2,Rize,Csize,Rowi,Rowj)
          <->(r2,  r3,   r9,  r10,  r4,  r5)
@ Copy row i of Mat1 to row j of Mat2
```

Example 2: Copy row i of Mat1 to row j of Mat2

C000	C004	C008	C00C	C010	C014	C018	C01C	C020	C024	C028	C02C	C030	C034	C038	C03C	C040	C044	C048	C04C	C050	C054
11	12	13	14	15	21	22	23	24	25	31	32	33	34	35	41	42	43	44	45		

R2

R6 = 5 x 2 = 10

R6 = 10 x 4= 40 = 0x28

row i=2

R2 = C000
R3 = C124
R4 = 2 ➔ row i
R5 = 3 ➔ row j
R9 = 4 ➔ # rows
R10 = 5 ➔ # cols

```
@ calculate byte offset to row i of Mat1
@ from array base as (Csize)x(rowi)x 4
   mul   r6,r10,r4     @ (Csize)x(Rowi)
   mov   r6,r6,LSL #2  @ x 4
   add   r2,r2,r6      @ r2=address of Mat1[i][0]
@ calculate byte offset to row j of Mat2
@ from array base as (Csize)x(Rowj)x 4
   mul   r6,r10,r5     @ (Csize)x(Rowj)
   mov   r6,r6,LSL #2  @ x 4
   add   r3,r3,r6      @ r3=address of Mat2[j][0]
Cprloop:
   ldr   r6,[r2],#4    @ get element from row i in Mat1
   str   r6,[r3],#4    @ store in row j in Mat2
   subs  r10,r10,#1 @ counter
   bne   Cprloop
```

Example 2: Copy row i of Mat1 to row j of Mat2

C000	C004	C008	C00C	C010	C014	C018	C01C	C020	C024	C028	C02C	C030	C034	C038	C03C	C040	C044	C048	C04C	C050	C054
11	12	13	14	15	21	22	23	24	25	31	32	33	34	35	41	42	43	44	45		

R6 = 10 x 4= 40 = 0x28

row i=2

R2 = C028
R3 = C124
R4 = 2 ➔ row i
R5 = 3 ➔ row j
R9 = 4 ➔ # rows
R10 = 5 ➔ # cols

```
@ calculate byte offset to row i of Mat1
@ from array base as (Csize)x(rowi)x 4
   mul   r6,r10,r4     @ (Csize)x(Rowi)
   mov   r6,r6,LSL #2  @ x 4
   add   r2,r2,r6      @ r2=address of Mat1[i][0]
@ calculate byte offset to row j of Mat2
@ from array base as (Csize)x(Rowj)x 4
   mul   r6,r10,r5     @ (Csize)x(Rowj)
   mov   r6,r6,LSL #2  @ x 4
   add   r3,r3,r6      @ r3=address of Mat2[j][0]
Cprloop:
   ldr   r6,[r2],#4    @ get element from row i in Mat1
   str   r6,[r3],#4    @ store in row j in Mat2
   subs  r10,r10,#1    @ counter
   bne   Cprloop
```

Example 2: Copy row i of Mat1 to row j of Mat 2

C124	C128	C12C	C130	C134	C138	C13C	C140	C144	C148	C14C	C150	C154	C158	C15C	C160	C164	C168	C16C	C170	C174	C178
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		

row j=3

R6 = 5 x 3 = 15

R6 = 15 x 4= 60 = 0x3C

R2 = C028
R3 = C124
R4 = 2 ➔ row i
R5 = 3 ➔ row j
R9 = 4 ➔ # rows
R10 = 5 ➔ # cols

```
@ calculate byte offset to row i of Mat1
@ from array base as (Csize)x(rowi)x 4
   mul   r6,r10,r4     @ (Csize)x(Rowi)
   mov   r6,r6,LSL #2  @ x 4
   add   r2,r2,r6      @ r2=address of Mat1[i][0]
@ calculate byte offset to row j of Mat2
@ from array base as (Csize)x(Rowj)x 4
   mul   r6,r10,r5     @ (Csize)x(Rowj)
   mov   r6,r6,LSL #2  @ x 4
   add   r3,r3,r6      @ r3=address of Mat2[j][0]
Cprloop:
   ldr   r6,[r2],#4    @ get element from row i in Mat1
   str   r6,[r3],#4    @ store in row j in Mat2
   subs  r10,r10,#1    @ counter
   bne   Cprloop
```

Example 2: Copy row i of Mat1 to row j of Mat 2

C124	C128	C12C	C130	C134	C138	C13C	C140	C144	C148	C14C	C150	C154	C158	C15C	C160	C164	C168	C16C	C170	C174	C178
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		

row j=3

R6 = 15 x 4= 60 = 0x3C

R2 = C028
R3 = C160
R4 = 2 ➔ row i
R5 = 3 ➔ row j
R9 = 4 ➔ # rows
R10 = 5 ➔ # cols

```
@ calculate byte offset to row i of Mat1
@ from array base as (Csize)x(rowi)x 4
   mul   r6,r10,r4     @ (Csize)x(Rowi)
   mov   r6,r6,LSL #2  @ x 4
   add   r2,r2,r6      @ r2=address of Mat1[i][0]
@ calculate byte offset to row j of Mat2
@ from array base as (Csize)x(Rowj)x 4
   mul   r6,r10,r5     @ (Csize)x(Rowj)
   mov   r6,r6,LSL #2  @ x 4
   add   r3,r3,r6      @ r3=address of Mat2[j][0]
Cprloop:
   ldr   r6,[r2],#4    @ get element from row i in Mat1
   str   r6,[r3],#4    @ store in row j in Mat2
   subs  r10,r10,#1    @ counter
   bne   Cprloop
```

C000	C004	C008	C00C	C010	C014	C018	C01C	C020	C024	C028	C02C	C030	C034	C038	C03C	C040	C044	C048	C04C	C050	C054
11	12	13	14	15	21	22	23	24	25	31	32	33	34	35	41	42	43	44	45		

R2

row i=2

C124	C128	C12C	C130	C134	C138	C13C	C140	C144	C148	C14C	C150	C154	C158	C15C	C160	C164	C168	C16C	C170	C174	C178
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		

R3

row j=3

R6 = 31

R2 = C028
R3 = C160
R10 = 5 → # cols

```
@ calculate byte offset to row i of Mat1
@ from array base as (Csize)x(rowi)x 4
   mul   r6,r10,r4     @ (Csize)x(Rowi)
   mov   r6,r6,LSL #2  @ x 4
   add   r2,r2,r6      @ r2=address of Mat1[i][0]
@ calculate byte offset to row j of Mat2
@ from array base as (Csize)x(Rowj)x 4
   mul   r6,r10,r5     @ (Csize)x(Rowj)
   mov   r6,r6,LSL #2  @ x 4
   add   r3,r3,r6      @ r3=address of Mat2[j][0]
Cprloop:
   ldr   r6,[r2],#4    @ get element from row i in Mat1
   str   r6,[r3],#4    @ store in row j in Mat2
   subs  r10,r10,#1    @ counter
   bne   Cprloop
```

C000	C004	C008	C00C	C010	C014	C018	C01C	C020	C024	C028	C02C	C030	C034	C038	C03C	C040	C044	C048	C04C	C050	C054
11	12	13	14	15	21	22	23	24	25	31	32	33	34	35	41	42	43	44	45		

R2

row i=2

C124	C128	C12C	C130	C134	C138	C13C	C140	C144	C148	C14C	C150	C154	C158	C15C	C160	C164	C168	C16C	C170	C174	C178
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		

R3

row j=3

```
@ calculate byte offset to row i of Mat1
@ from array base as (Csize)x(rowi)x 4
   mul   r6,r10,r4     @ (Csize)x(Rowi)
   mov   r6,r6,LSL #2  @ x 4
   add   r2,r2,r6      @ r2=address of Mat1[i][0]
@ calculate byte offset to row j of Mat2
@ from array base as (Csize)x(Rowj)x 4
   mul   r6,r10,r5     @ (Csize)x(Rowj)
   mov   r6,r6,LSL #2  @ x 4
   add   r3,r3,r6      @ r3=address of Mat2[j][0]
Cprloop:
   ldr   r6,[r2],#4    @ get element from row i in Mat1
   str   r6,[r3],#4    @ store in row j in Mat2
   subs  r10,r10,#1 @ counter
   bne   Cprloop
```

R6 = 31

R2 = C02C
R3 = C160
R10 = 5 ➔ # cols

C000	C004	C008	C00C	C010	C014	C018	C01C	C020	C024	C028	C02C	C030	C034	C038	C03C	C040	C044	C048	C04C	C050	C054
11	12	13	14	15	21	22	23	24	25	31	32	33	34	35	41	42	43	44	45		

R2

row i=2

C124	C128	C12C	C130	C134	C138	C13C	C140	C144	C148	C14C	C150	C154	C158	C15C	C160	C164	C168	C16C	C170	C174	C178
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	31	0	0	0	0		

row j=3

R3

R6 = 31

```
@ calculate byte offset to row i of Mat1
@ from array base as (Csize)x(rowi)x 4
   mul   r6,r10,r4     @ (Csize)x(Rowi)
   mov   r6,r6,LSL #2 @ x 4
   add   r2,r2,r6      @ r2=address of Mat1[i][0]
@ calculate byte offset to row j of Mat2
@ from array base as (Csize)x(Rowj)x 4
   mul   r6,r10,r5     @ (Csize)x(Rowj)
   mov   r6,r6,LSL #2 @ x 4
   add   r3,r3,r6      @ r3=address of Mat2[j][0]
Cprloop:
   ldr   r6,[r2],#4    @ get element from row i in Mat1
   str   r6,[r3],#4    @ store in row j in Mat2
   subs r10,r10,#4     @ counter
   bne   Cprloop
```

R2 = C02C
R3 = C160
R10 = 5 ➔ # cols

C000	C004	C008	C00C	C010	C014	C018	C01C	C020	C024	C028	C02C	C030	C034	C038	C03C	C040	C044	C048	C04C	C050	C054
11	12	13	14	15	21	22	23	24	25	31	32	33	34	35	41	42	43	44	45		

R2

row i=2

C124	C128	C12C	C130	C134	C138	C13C	C140	C144	C148	C14C	C150	C154	C158	C15C	C160	C164	C168	C16C	C170	C174	C178
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	31	0	0	0	0		

row j=3

R3

R6 = 31

```
@ calculate byte offset to row i of Mat1
@ from array base as (Csize)x(rowi)x 4
   mul   r6,r10,r4     @ (Csize)x(Rowi)
   mov   r6,r6,LSL #2  @ x 4
   add   r2,r2,r6      @ r2=address of Mat1[i][0]
@ calculate byte offset to row j of Mat2
@ from array base as (Csize)x(Rowj)x 4
   mul   r6,r10,r5     @ (Csize)x(Rowj)
   mov   r6,r6,LSL #2  @ x 4
   add   r3,r3,r6      @ r3=address of Mat2[j][0]
Cprloop:
   ldr   r6,[r2],#4    @ get element from row i in Mat1
   str   r6,[r3],#4    @ store in row j in Mat2
   subs  r10,r10,#1    @ counter
   bne   Cprloop
```

R2 = C02C
R3 = C164
R10 = 5 ➔ # cols

C000	C004	C008	C00C	C010	C014	C018	C01C	C020	C024	C028	C02C	C030	C034	C038	C03C	C040	C044	C048	C04C	C050	C054
11	12	13	14	15	21	22	23	24	25	31	32	33	34	35	41	42	43	44	45		

R2

row i=2

C124	C128	C12C	C130	C134	C138	C13C	C140	C144	C148	C14C	C150	C154	C158	C15C	C160	C164	C168	C16C	C170	C174	C178
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	31 0	0	0	0	0		

row j=3

R3

```
@ calculate byte offset to row i of Mat1
@ from array base as (Csize)x(rowi)x 4
   mul   r6,r10,r4     @ (Csize)x(Rowi)
   mov   r6,r6,LSL #2  @ x 4
   add   r2,r2,r6      @ r2=address of Mat1[i][0]
@ calculate byte offset to row j of Mat2
@ from array base as (Csize)x(Rowj)x 4
   mul   r6,r10,r5     @ (Csize)x(Rowj)
   mov   r6,r6,LSL #2  @ x 4
   add   r3,r3,r6      @ r3=address of Mat2[j][0]
Cprloop:
   ldr   r6,[r2],#4    @ get element from row i in Mat1
   str   r6,[r3],#4    @ store in row j in Mat2
   subs  r10,r10,#4    @ counter
   bne   Cprloop
```

R2 = C02C
R3 = C164
R10 = 5 ➔ # cols

C000	C004	C008	C00C	C010	C014	C018	C01C	C020	C024	C028	C02C	C030	C034	C038	C03C	C040	C044	C048	C04C	C050	C054
11	12	13	14	15	21	22	23	24	25	31	32	33	34	35	41	42	43	44	45		

row i=2

C124	C128	C12C	C130	C134	C138	C13C	C140	C144	C148	C14C	C150	C154	C158	C15C	C160	C164	C168	C16C	C170	C174	C178
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		

row j=3

```
@ calculate byte offset to row i of Mat1
@ from array base as (Csize)x(rowi)x 4
   mul   r6,r10,r4     @ (Csize)x(Rowi)
   mov   r6,r6,LSL #2  @ x 4
   add   r2,r2,r6      @ r2=address of Mat1[i]
@ calculate byte offset to row j of Mat2
@ from array base as (Csize)x(Rowj)x 4
   mul   r6,r10,r5     @ (Csize)x(Rowj)
   mov   r6,r6,LSL #2  @ x 4
   add   r3,r3,r6      @ r3=address of Mat2[j]
Cprloop:
   ldr   r6,[r2],#4    @ get element from row    n Mat1
   str   r6,[r3],#4    @ store in row j in Mat
   subs  r10,r10,#4    @ counter
   bne   Cprloop
```

DO YOU WANT TO LEARN?

go through the code manually step by step

```
@ ****   Fibonacci Sequence     ****
@ This program produces the first 10 Fibonacci numbers ;
@ and their sum
@ It shows a loop structure (DO..WHILE)
@ Init:     PrevFib = 1
@           CurrFib = 2
@           TotSum = 3
@           Count = 8
@ Body:     NextFib = PrevFib + CurrFib
@           TotSum = TotSum + NextFib
@           PrevFib = CurrFib
@           CurrFib = NextFib
@ Decr:     Count = Count -1
@ Test:     IF Count > 0 goto Body
@           ELSE Print TotSum
@ Register usage:
@ PrevFib <-> R1
@ CurrFib <-> R2
@ NextFib <-> R3
@ TotSum <-> R4
@ Count <-> R5
```

Study by yourself

```
        .text
        .global _start
        .equ   MAX,10
        .equ   EXIT,0x11
_start:
        mov    r1,#1          @ PrevFib = 1
        mov    r2,#2          @ CurrFib = 2
        mov    r4,#3          @ TotSum = 3
        mov    r5,#MAX        @ Count = MAX-2
        sub    r5,r5,#2

Body:   add    r3,r1,r2       @ NextFib = PrevFib + CurrFib
        add    r4,r4,r3       @ TotSum = TotSum + NextFib
        mov    r1,r2          @ PrevFib = CurrFib
        mov    r2,r3          @ CurrFib = NextFib

Decr:   subs   r5,r5,#1       @ Count = Count -1
        bne    Body           @ If Count != 0, repeat loop
Done:   swi    EXIT
        .end
```

Study by yourself

```
@ ****   Fibonacci Sequence     ****
@ This program produces the first 10 Fibonacci numbers and their sum
@ It shows a loop structure (DO..WHILE)
@ Init:  PrevFib = 1; CurrFib = 2; TotSum = 3; Count = 8
@ Body:  NextFib = PrevFib + CurrFib
@        TotSum = TotSum + NextFib
@        PrevFib = CurrFib
@        CurrFib = NextFib
@ Decr:  Count = Count -1
@ Test:  IF Count > 0 goto Body ELSE Print TotSum
@ Register usage: PrevFib <-> R1; CurrFib <-> R2; NextFib <-> R3
@ TotSum <-> R4; Count <-> R5
        .text
        .global _start
        .equ    MAX,10
        .equ    EXIT,0x11
_start:
        mov     r1,#1           @ PrevFib = 1
        mov     r2,#2           @ CurrFib = 2
        mov     r4,#3           @ TotSum = 3
        mov     r5,#MAX  @ Count = MAX-2
        sub     r5,r5,#2
Body:   add     r3,r1,r2 @ NextFib = PrevFib + CurrFib
        add     r4,r4,r3 @ TotSum = TotSum + NextFib
        mov     r1,r2           @ PrevFib = CurrFib
        mov     r2,r3           @ CurrFib = NextFib
Decr:   subs    r5,r5,#1 @ Count = Count -1
        bne     Body            @ If Count != 0, repeat loop
Done:   swi     EXIT
        .end
```

Study by yourself

Some Advice on Assembly Language Programming

- you are responsible for ensuring clear control structures
- you are responsible for the assignment of memory
- you are responsible for ensuring the correct and consistent interpretation of data
- labels have fixed values – they identify a location in memory which never moves during program execution
- labels are not variables – they do not contain values
- an assembler program is syntactically one unit – labels are accessible throughout the whole program so ***labels must be unique across the whole program***

```
@ *** Check for palindrome in a binary number ***
@ to show a couple of shifting operations
      .text
      .global _start
      .equ  BINNUM,0x80000001
      .equ  LENGTH,16
_start:
      mov   r0,#0         @set for palindrome = no
      ldr   r1,=BINNUM    @number to be checked
      mov   r2,#0         @used to construct shifted value
      mov   r3,#LENGTH    @loop count=16 for 32 bit number
loop: movs  r1,r1,lsl #1        @shift 1 left into carry
      movs  r2,r2,rrx           @rotate 1 in from carry
      subs  r3,r3,#1            @decrement loop counter
      bne   loop
      cmp   r1,r2
      bne   done
      mov   r0,#1         @if equal, set flag for yes
                          @to palindrome
done: swi   0x11
      .end
```

Study by yourself - advanced!