CSE 211: Introduction to Embedded Systems

Lab 2

Lab2 Agenda:

- Reset vectors.
- Solve Lab problems 1 and 2

Reset

- Our *single file* program is a valid **minimal** program.
  (not because it has only a few instructions, but because it contains only the *necessary vectors* (lines 6 and 7) for the processor to begin running).
- A non-minimal program has its full **vector table** in a typical **startup** file.
- Reset vector table is mapped to address 0 in ROM.
- Line 6 contains the address (0x20008000) of the **stack pointer** when the stack is empty.
- Line 7 contains the label where code instructions start.

lab.s

```
1          ; Vector Table Mapped to Address 0 at Reset
2          ; Linker requires __Vectors to be exported
3          AREA RESET, DATA, READONLY
4          EXPORT  __Vectors           ; to be used externally by linker
5  __Vectors
6          DCD     0x20008000          ; initial stack pointer
7          DCD     Reset_Handler       ; reset vector
8          ALIGN
```

Reset

- reset occurs immediately after power is applied and when the reset signal is asserted (using the reset button).
- @ reset

lab.s

```
        ; Vector Table Mapped to Address 0 at Reset
        ; Linker requires __Vectors to be exported
        AREA RESET, DATA, READONLY
        EXPORT  __Vectors           ; to be used externally by linker
__Vectors
        DCD     0x20008000          ; initial stack pointer
        DCD     Reset_Handler       ; reset vector
        ALIGN
```

The 32-bit value at flash ROM location **0** is loaded into the **SP**.

the 32-bit value at location 4 is loaded into the PC

Lab 2: Q1

- Write ARM assembly code to sum the array items of size 10.

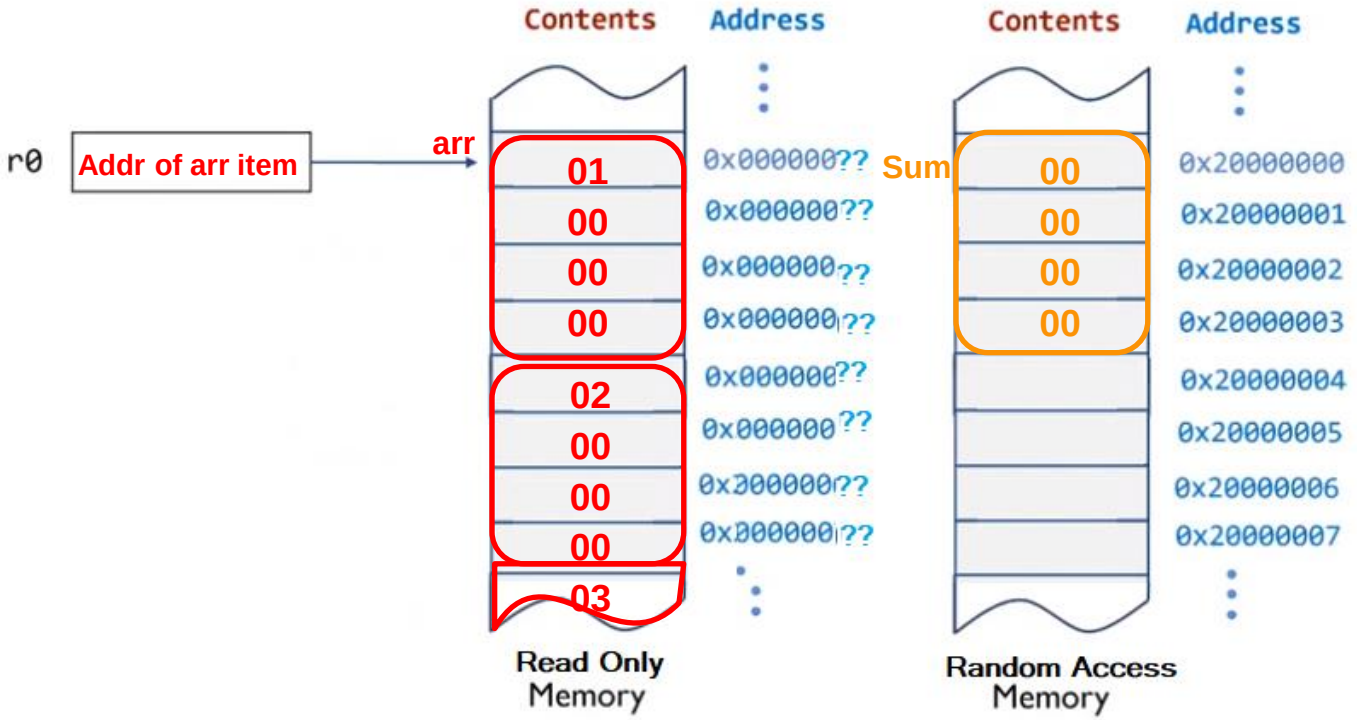
  The array contains the following values:

  1, 2, 3, 4, 5, 6, 7,8, 9, 10.

```
int arr [10] = {1,2,3,4,5,6,7,8,9,10};
int n=10;
int sum = 0;
while (n>0)
{
	sum+= arr[n-1] ;
	n--;
}
```

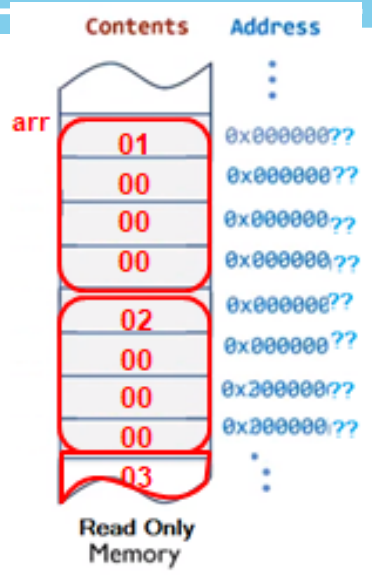
Lab 2: Q1

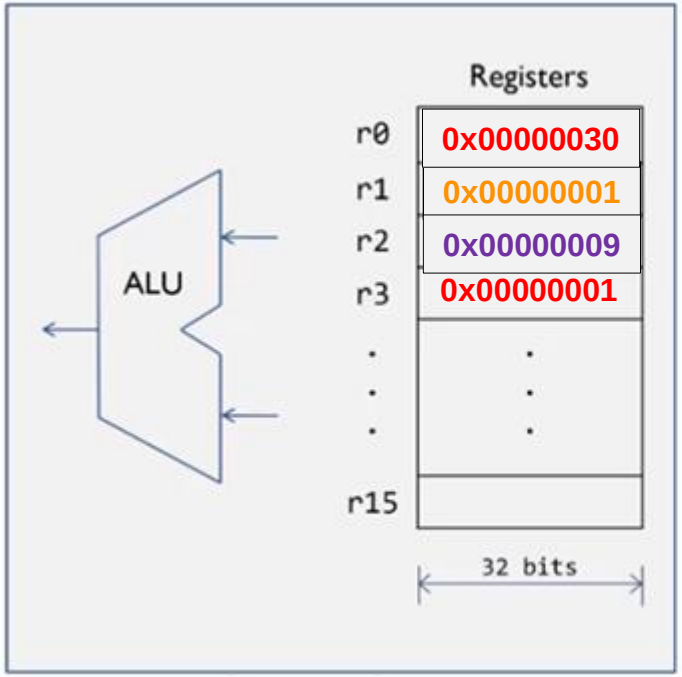
```
        ALIGN
        AREA myConstData, CODE, READONLY
arr     DCD 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

        ALIGN
        AREA myVarData, DATA, READWRITE
SUM     DCD 0
```



```
        AREA myCode, CODE, ReadOnly
        ENTRY
        EXPORT Reset_Handler

Reset_Handler
        LDR  R0, =arr
        MOV R1,#0
        MOV R2,#10

LOOP    LDR R3, [R0], #4
        ADD R1, R1, R3
        SUBS R2, R2, #1
        BNE LOOP
```

```
int arr [10] = {1,2,3,4,5,6,7,8,9,10};
int n=10;
int sum = 0;
while (n>0)
{
    sum+= arr[n-1] ;
    n--;
}
```



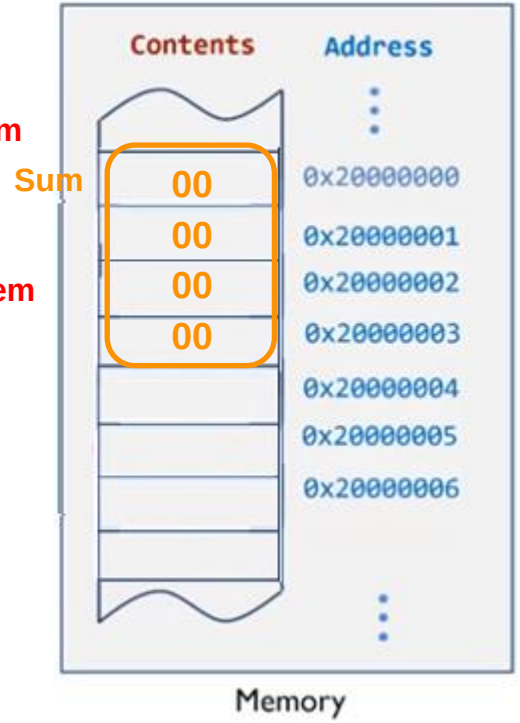


Addr of arr item

Sum value

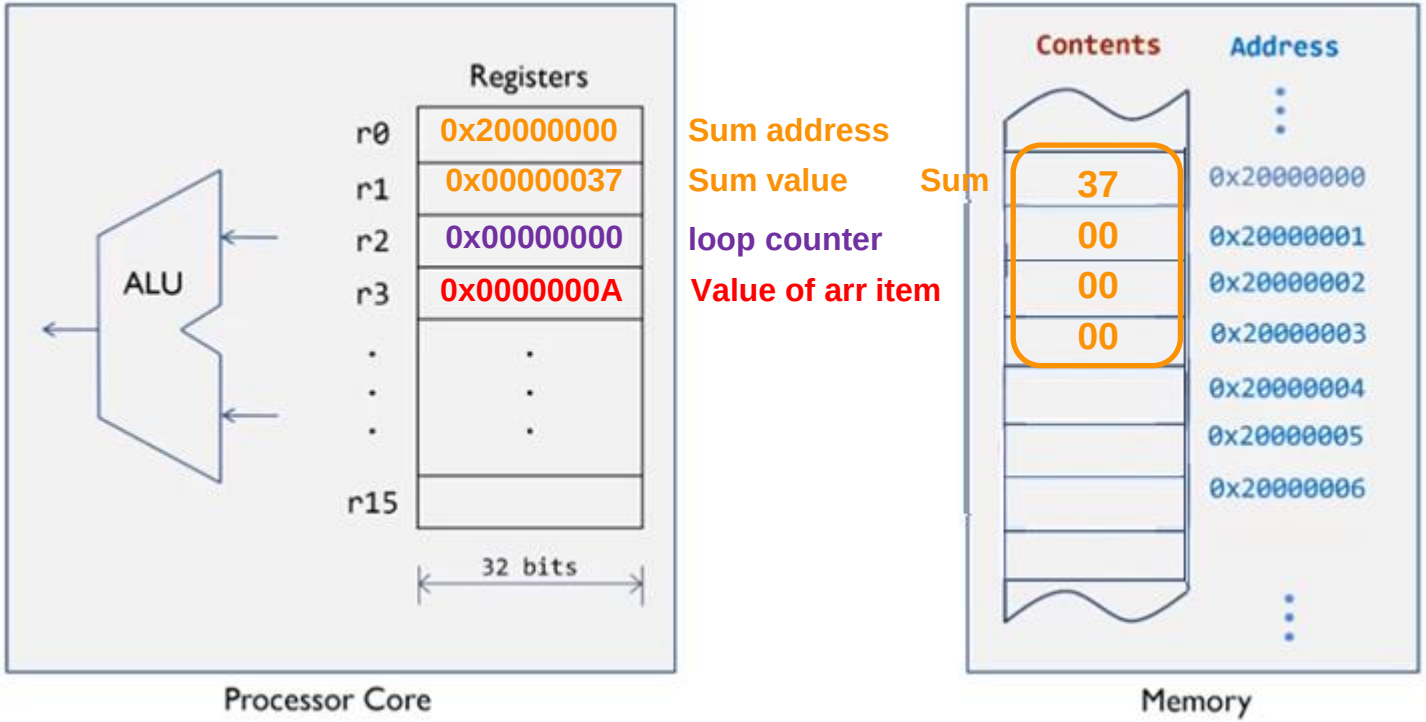
loop counter

Value of arr item



```
        AREA myCode, CODE, ReadOnly
        ENTRY
        EXPORT Reset_Handler

Reset_Handler
        LDR  R0, =arr
        MOV R1,#0
        MOV R2,#10

LOOP    LDR R3, [R0], #4
        ADD R1, R1, R3
        SUBS R2, R2, #1
        BNE LOOP

        LDR R0, =sum
        STR R1, [R0]
```

```
int arr [10] = {1,2,3,4,5,6,7,8,9,10};
int n=10;
int sum = 0;
while (n>0)
{
    sum+= arr[n] ;
    n--;
}
```



Lab2: Q1 Answer

```
        AREA myCode, CODE, ReadOnly
        ENTRY
        EXPORT Reset_Handler

Reset_Handler

ARR_SIZE    EQU     10
        LDR R0, =arr
        MOV R1, #0          ; R1 for sum
        MOV R2, #ARR_SIZE   ; R2 for loop counter (ARR_SIZE)
L1
        LDR R3, [R0], #4    ; load a number into R3 and increment the offset
        ADD R1, R1, R3      ; R1 += R3
        SUBS R2, R2, #1     ; R2 --
        BNE L1              ; loop until R2 == 0

        LDR R0, =SUM        ; R1 contains the sum
        STR R1, [R0]        ; store it in the SUM variable

dloop   b dloop

        END
```

Lab 2: Q2

- Assume A is a label for 4x4 matrix and

  Z, and X are labels for arrays with 4 items (each item is 4 bytes)

  Write arm assembly for the following snippet code.

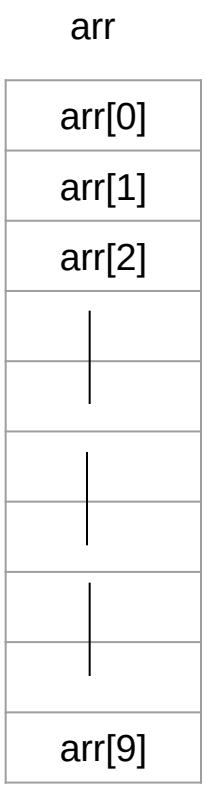
```
for (int row = 0; row < 4; row++)
    for (int column = 0; column < 4; column++)
        Z[row] += A [row, column] * X[column]
```

1D array

C code

To access general element use

arr[i]



Logical structure of array **arr**

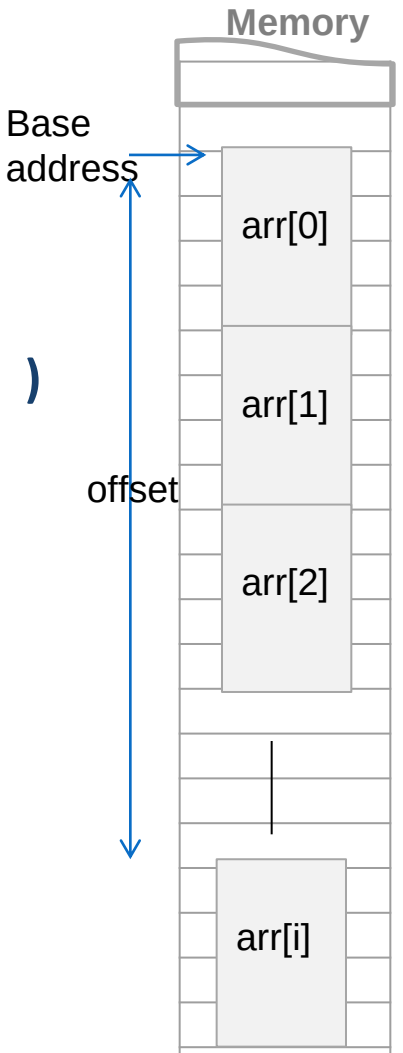
Assembly code

To access general element use

Base address + offset

= Base address + (No. of preceding items * item size)

= Base address + i*4



1D array

C code

```c
int arr[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
int sum = 0 ;
for(int i=0 ; i<10 ; i++)
    sum+= arr[i];
```

Assembly code

```
Reset_Handler

ARR_SIZE    EQU     10
        LDR R0, =arr
        MOV R1, #0          ; R1 for sum
        MOV R2, #ARR_SIZE   ; R2 for (ARR_SIZE)
        MOV R4, #0          ; R4 for loop counter
L1
        LDR R3, [R0, R4, LSL #2 ]   ; load an item arr[i] into R3
        ADD R1, R1, R3      ; R1+= R3   sum+=arr[i]
        ADD R4, R4, #1      ; R4++      i++
        CMP R4, R2
        BNE L1              ; loop until R2 == 0

        LDR R0, =SUM        ; R1 contains the sum
        STR R1, [R0]        ; store it in the SUM variable

dloop   b dloop

        END
```

Base address

Offset items*4

2D array

C code

To access general element in mxn matrix use

A[i][j]
=A[row][column]

0 <= row < m
0 <= column < n

A

A[0][0]	A[0][1]	A[0][2]	A[0][3]
A[1][0]	A[1][1]	A[1][2]	A[1][3]
A[2][0]	A[2][1]	A[2][2]	A[2][3]
A[3][0]	A[3][1]	A[3][2]	A[3][3]

Assembly code

To access general element use

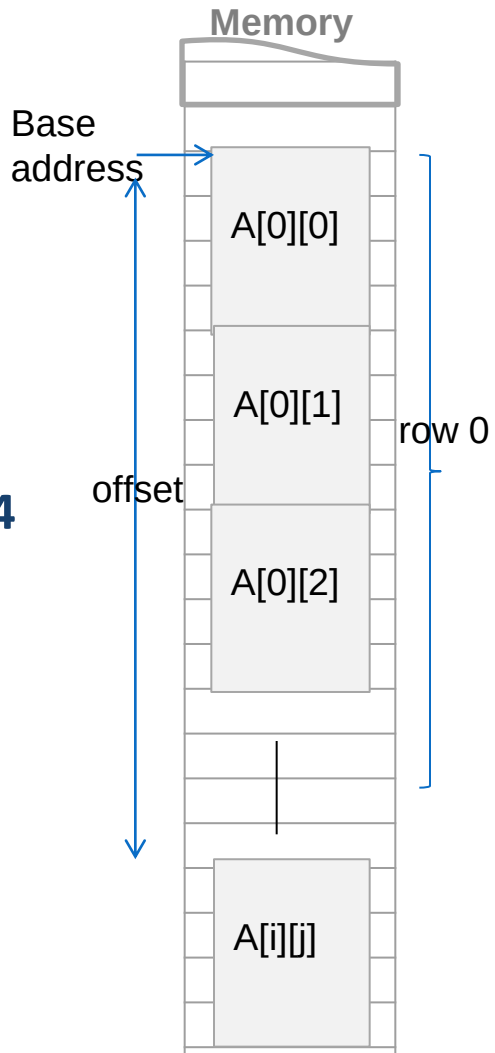
Base address + offset

= Base address + No. of preceding items * item size

= Base address +(row*row_size + column)*4

Rows counter

Columns counter



2D array

C code

```
int A[4][4] = {{1, 2, 3, 4},{ 5, 6, 7, 8},
                { 9, 10, 11, 12} , {13, 14, 15, 16}};
int sum = 0 ;
for(int i=0 ; i<4 ; i++)
    for (int j=0 ; j<4 ; j++)
        sum+= A[i][j];
```

Assembly code

```
Reset_Handler

SIZE    EQU     4
        LDR R0, =A
        MOV R1, #0          ; R1 for sum
        MOV R2, #SIZE       ; R2 for (ARR_SIZE)
        MOV R4, #0          ; R4 for loop counter i
L       MOV R5, #0          ; R4 for loop counter j
L1
        MUL R6, R4, R2              ; i* row size
        ADD R6 , R6, R5             ; offset = (i* row size) + j
        LDR R3, [R0, R6, LSL #2 ]   ; load an item A[i][j] into R3
        ADD R1, R1, R3      ; R1+= R3   sum+=A[i][j]
        ADD R5, R5, #1      ; R5++      j++
        CMP R5, R2
        BNE L1              ; loop until R5 == 4

        ADD R4, R4, #1      ; R4++      i++
        CMP R5, R2
        BNE L

        LDR R0, =SUM        ; R1 contains the sum
        STR R1, [R0]        ; store it in the SUM variable

dloop   b dloop

        END
```

R6= no of preceding items

Multiply matrix by vector in C

A

	col 0	col 1	col 2	col 3
row 0	A[0][0]	A[0][1]	A[0][2]	A[0][3]
row 1	A[1][0]	A[1][1]	A[1][2]	A[1][3]
row 2	A[2][0]	A[2][1]	A[2][2]	A[2][3]
row 3	A[3][0]	A[3][1]	A[3][2]	A[3][3]

mxn

X

x

x[0]
x[1]
x[2]
x[3]

nx1

=

z

z[0]
z[1]
z[2]
z[3]

mx1

Z[1] = A[1][0].x[0] + A[1][1].x[1] + A[1][2].x[2] + A[1][3].x[3]

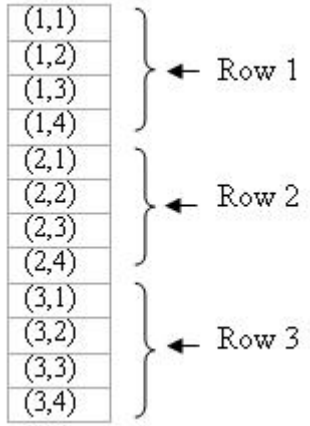
Z[row] += A[row][column].x[column]

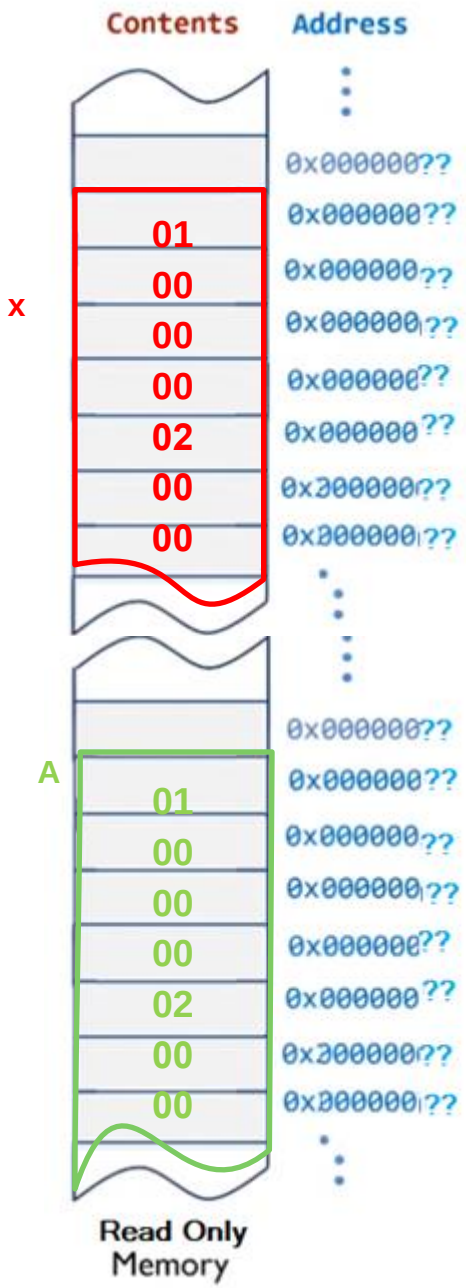
```
for (int row = 0; row < 4; row++)
    for (int column = 0; column < 4; column++)
        Z[row] += A [row, column] * X[column]
```

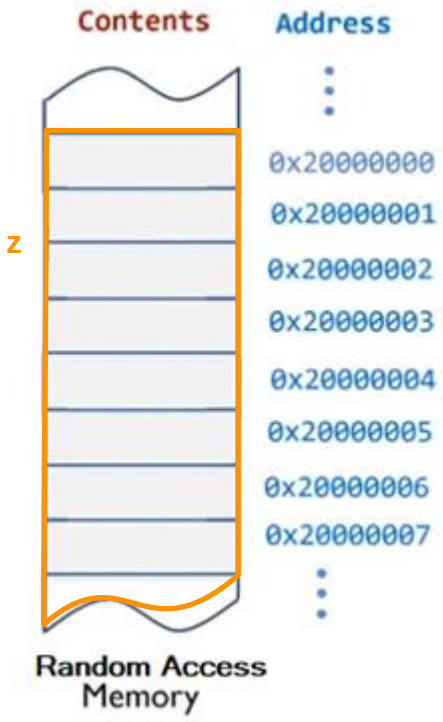
Lab 2: Q2

```
        ALIGN
        AREA myConstData, CODE, READONLY

X       DCD 1, 2, 3, 4
A       DCD 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16   ✓

        ALIGN
        AREA myVarData, DATA, READWRITE
Z       SPACE 16
```

Row major order







```
Reset_Handler
SIZE        EQU     4

                MOV R0, #0                  ; row=0
                MOV R1, #0                  ; column=0
                MOV R2, #SIZE               ; Size=4
                LDR R3, =A                  ; R3 = base address of A
                LDR R4, =X                  ; R4 = base address of X
                LDR R5, =Z                  ; R5 = base address of Z
OuterLoop
                CMP R0, #SIZE               ;row < 4
                BEQ ENDOuterLoop
                MOV R1, #0                  ; column=0
                MOV R8, #0                  ;Z[row]=0
InnerLoop
                CMP R1, R2                  ; column< 4
                BEQ EndInnerLoop
                LDR R6,[R4,R1, LSL #2]      ; load X[column]
                MUL R9,R0, #SIZE            ; row*4
                ADD R9,R9,R1                ; row*4+column
                LDR R7,[R3,R9, LSL #2]      ; load A[row][column]
                MUL R10,R7,R6               ; A[row][column] * X[column]
                ADD R8, R8, R10             ; Z[row]+= A[row][column] * X[column]
                ADD R1,R1,#1                ; column=column+1
                B InnerLoop
EndInnerLoop    STR R8,[R5,R0, LSL #2]      ; store result R11 into Z[row]
                ADD R0,R0,#1                ; row=row+1
                B       OuterLoop
ENDOuterLoop
                END
```

```
for(int row=0 ; row<size ; row++)
    z[row] = 0 ;
    (for int column=0 ; column<size ; column++)
        Z[row]+= A[row][column] * X[column]
```

```
; R0 row
; R1 column
; R2 Size=4
; R3 base address of A
; R4 base address of X
; R5 base address of Z
; R6 X[column]
; R7 A[row][column]
; R8 Z[row]
; R9 [(row*row_size)+column)]
; R10  A[row][column] * X[column]
```

Addressing Modes

Addressing without Offset

```
LDR r1, [r0]          ; r0 holds the memory address
```

Addressing with Offset (*Offset*: range -255 to + **4095**)

```
LDR r1, [r0,  #4]     ; pre-index
                      ; r1 =word pointed to by r0+4

LDR r1, [r0], #4      ; post-index
                      ; r1 = word pointed to by r0, then r0=r0+4

LDR r1, [r0,  #4]!    ; pre-index with update
                      ;  first r0=r0+4,  then r1 = word pointed to by r0

LDR r1, [r0,  r2 ]    ; pre-index
                      ; r1 =word pointed to by r0+ r2

LDR r1, [r0, r2, LSL #2]; pre-index
                      ; r1 =word pointed to by r0+ 4*r2
```

Updating NZCV flags in PSR

Flags not changed		Flags updated
ADD	→	ADDS
SUB	→	SUBS
MUL	→	MULS
UDIV	→	UDIVS
AND	→	ANDS
ORR	→	ORRS
LSL	→	LSLS
MOV	→	MOVS

Most instructions update NZCV flags only if S suffix is present

Some instructions update NZCV flags even if no S is specified.

- CMP: Compare, like SUBS but without destination register
- CMN: Compare Negative, like ADDS but without destination register

Suffix	Description	Flags tested
EQ	EQual	Z == 1
NE	Not Equal	Z == 0
CS/HS	Unsigned Higher or Same	C == 1
CC/LO	Unsigned LOwer	C == 0
MI	MInus / Negative	N == 1
PL	PLus / Positive or Zero	N == 0
VS	oVerflow Set	V == 1
VC	oVerflow Cleared	V == 0
HI	Unsigned HIgher	C == 1 & Z == 0
LS	Unsigned Lower or Same	C == 0 or Z == 1
GE	Signed Greater or Equal	N == V
LT	Signed Less Than	N != V
GT	Signed Greater Than	Z == 0 & N == V
LE	Signed Less than or Equal	Z == 1 or N != V

Assembler Directives

- We use assembler directives to assist and control the assembly process.
- Directives or pseudo-ops are not part of the instruction set. These directives change the way the code is assembled.
- They tell the assembler the organization of the source code, but they are not code and are not in the final executable.

Assembler Directives

- **Thumb** - placed at the top of the file to specify that code is generated with Thumb instructions.
- **CODE** - Denotes the section for machine instructions (ROM).
- **DATA** - Denotes the section for global variables(RAM).
- **AREA** -Instructs the assembler to assemble a new code or data section.
- **SPACE** – Reserves a block of memory and fills it with zeros.
- **ALIGN** – Used to ensure next object aligns properly.
- **[.text]** - Makes assembly code callable by C

Assembler Directives

- **EXPORT** – to make an object accessible from another file
- **IMPORT** – to access an "exported" object
- **END** - Placed at the end of each file
- **DCB** – Places byte (8-bits) sized constant in memory
- **DCW**- Places a half-word(16-bits) sized constant into memory.
- **DCD** - Places a word (32-bits) sized constant into memory.
- **EQU** – To give a symbolic name to a numeric constant.

Thanks