Arrays and Structures (7A)

Young Won Lim
3/11/20

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using LibreOffice.

Young Won Lim
3/11/20

Based on

ARM System-on-Chip Architecture, 2nd ed, Steve Furber

Introduction to ARM Cortex-M Microcontrollers
– Embedded Systems, Jonathan W. Valvano

Digital Design and Computer Architecture,
D. M. Harris and S. L. Harris

Computer Organization and Design ARM Edition:
The Hardware Software Interface
By David A. Patterson, John L. Hennessy

https://thinkingeek.com/arm-assembler-raspberry-pi/

Memory allocating directives

DCW	Word	allocates Halfwords	2 bytes	
DCD	Double Word	allocates Words	4 bytes	&
DCB	Byte	allocates Bytes	1 byte	=
SPACE		allocates a number of zeroed bytes		%
FILL		allocates a number of filled bytes		
EQU		defines a constant		*
RN		aliased registers		

http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0489c/Cacbbebi.html

DCW, DCWU

The DCW directive allocates one or more halfwords of memory,
aligned on two-byte boundaries, and defines
the initial runtime contents of the memory.

DCW inserts a byte of padding before the first defined halfword
if necessary to achieve two-byte alignment.

Use DCWU if you do not require alignment.

```
Data    DCW     -225,2*number   ; number must already be defined
        DCWU    number+4
```

http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0489c/Cacbbebi.html

DCD, DCDU

The DCD directive allocates one or more words of memory,
aligned on four-byte boundaries, and defines
the initial runtime contents of the memory.
& is a synonym for DCD.

DCD inserts up to three bytes of padding before the first defined word,
if necessary, to achieve four-byte alignment.

Use DCDU if you do not require alignment.

```
Data1   DCD     1,5,20          ; Defines 3 words containing
                                ; decimal values 1, 5, and 20
Data2   DCD     mem06 + 4       ; Defines 1 word containing 4 +
                                ; the address of the label   mem06
        AREA    MyData, DATA, READWRITE
        DCB     255             ; Now misaligned ...
Data3   DCDU    1,5,20          ; Defines 3 words containing
                                ; 1, 5 and 20, not word aligned
```

http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0489c/Cacbbebi.html

DCB

The DCB directive allocates one or more bytes of memory,
and defines the initial runtime contents of the memory.
= is a synonym for DCB.

a numeric expression that evaluates to an integer in the range -128 to 255.
a quoted string.

If DCB is followed by an instruction,
use an ALIGN directive to ensure that the instruction is aligned.

Unlike C strings, ARM assembler strings are not nul-terminated.
You can construct a nul-terminated C string using DCB as follows:

```
C_string    DCB     "C_string",0
```

http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0489c/Cacbbebi.html

SPACE, FILL

The **SPACE** directive reserves a zeroed block of memory.
% is a synonym for **SPACE**.

The **FILL** directive reserves a block of memory to fill with the given value.

Use the **ALIGN** directive to align any code following a SPACE or FILL directive.

```
        AREA    MyData, DATA, READWRITE
data1   SPACE   255         ; defines 255 bytes of zeroed store
data2   FILL    50,0xAB,1 ; defines 50 bytes containing 0xAB
```

http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0489c/Cacbbebi.html

EQU

The **EQU** directive gives a symbolic name
to a numeric constant,
a register-relative value or
a PC-relative value.
***** is a synonym for **EQU**.

Use **EQU** to define constants.
This is similar to the use of #define to define a constant in C.

```
abc EQU 2               ; assigns the value 2 to the symbol abc.
xyz EQU label+8         ; assigns the address (label+8) to the
                        ; symbol xyz.
fiq EQU 0x1C, CODE32    ; assigns the absolute address 0x1C to
                        ; the symbol fiq, and marks it as code
```

http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0489c/Cacbbebi.html

RN

Use **RN** to allocate convenient names to registers,
to help you to remember what you use each register for.

Be careful to avoid conflicting uses of the same register under different names.

```
regname     RN  11  ; defines regname for register 11
sqr4        RN  r6  ; defines sqr4 for register 6
```

http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0489c/Cacbbebi.html

BX (Branch and Exchange)

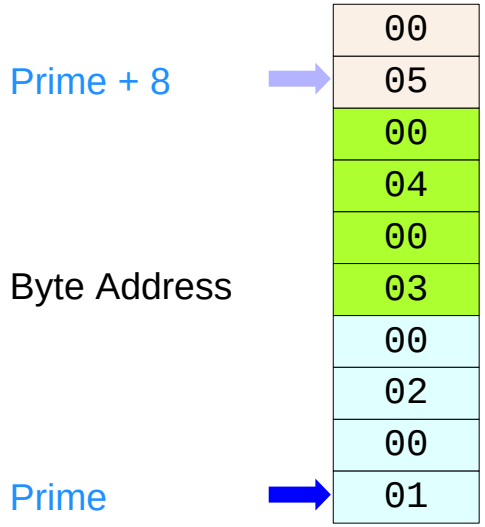
The BX instruction causes a branch to the address contained in Rm
and exchanges the instruction set, if required:

BX Rm derives the target instruction set from bit[0] of Rm:
If bit[0] of Rm is 0, the processor changes to, or remains in, ARM state.
If bit[0] of Rm is 1, the processor changes to, or remains in, Thumb state.

http://www.keil.com/support/man/docs/armasm/armasm_dom1361289866466.htm

Initialized Array

```
unit16_t  const  Prime[5] = {1, 2, 3, 4, 5};


Prime  DCW    1, 2, 3, 4, 5

LDR    R1, =Prime            ; pointer to the structures
LDRH   R0, [R1, #8]          ; read 16-bit unsigned Prime[4] (= 5)
```

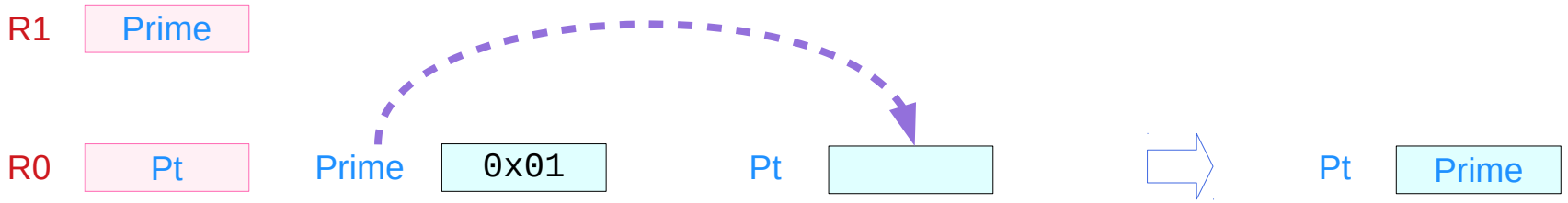


Word Address

Prime / 4 + 2	–	–	00	05
	00	04	00	03
Prime / 4	00	02	00	01

Pointer access to an array

```
unit16_t  const  *Pt;

Pt = Prime;
Pt = &Prime[0];


Pt      SPACE  4

LDR     R1,=Prime          ; pointer to the structure
LDR     R0,=Pt             ; pointer to the Pt
STR     R1,[R0]            ; Pt is a pointer to Prime[0]
```



Introduction to ARM Cortex-M Microcontrollers – Embedded Systems, Jonathan W. Valvano

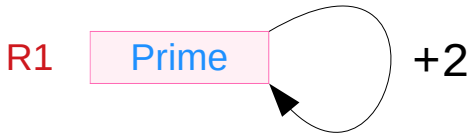
Pointer access to an array

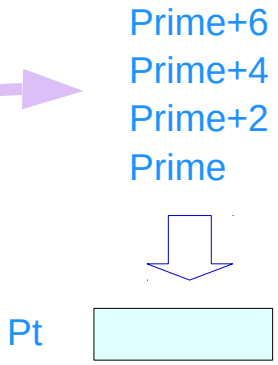
```
unit16_t  const  *Pt;

Pt = Prime;
Pt = &Prime[0];
```

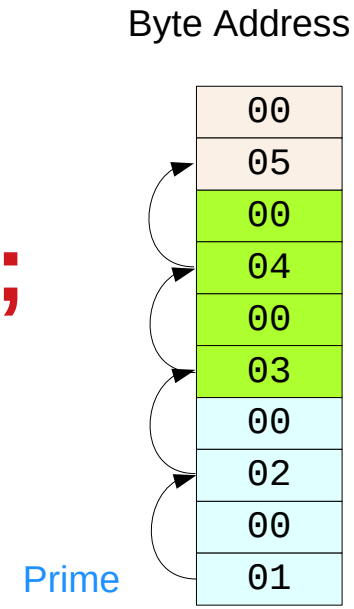
Pt++;

Pt = Pt + 1;

```
Pt      SPACE  4

LDR     R0, =Pt
LDR     R1, [R0]
ADD     R1, #2
STR     R1, [R0]
```







Introduction to ARM Cortex-M Microcontrollers – Embedded Systems, Jonathan W. Valvano

Pointer access to an array

```
unit8_t   VSuffer[100];
unit8_t   PSuffer[100];


Vbuffer   SPACE  100    ; 100 zeroed bytes
Pbuffer   SPACE  100    ; 100 zeroed bytes
```

Pointer access to an array

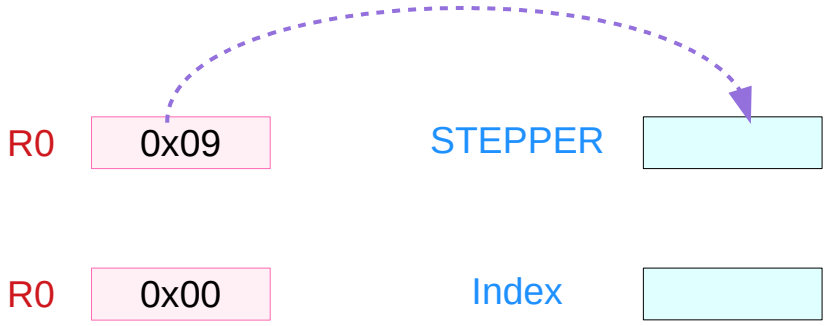
```
void Sort(unit8_t *bufPt) {  // example key operations in sort
   unit8_t data;
   data = *bufPt;   // example read data from original buffer
   *bufPt = data;   // example write data back t the buffer
}

void main(void) {
   Sort(VBuffer);
   Sort(PBuffer);
}
```

```
LDR    R0, =VBuffer      ; R0 = &VBuffer[0]
BL     Sort

LDR    R0, =PBuffer      ; R0 = &VBuffer[0]
BL     Sort
```

Introduction to ARM Cortex-M Microcontrollers – Embedded Systems, Jonathan W. Valvano

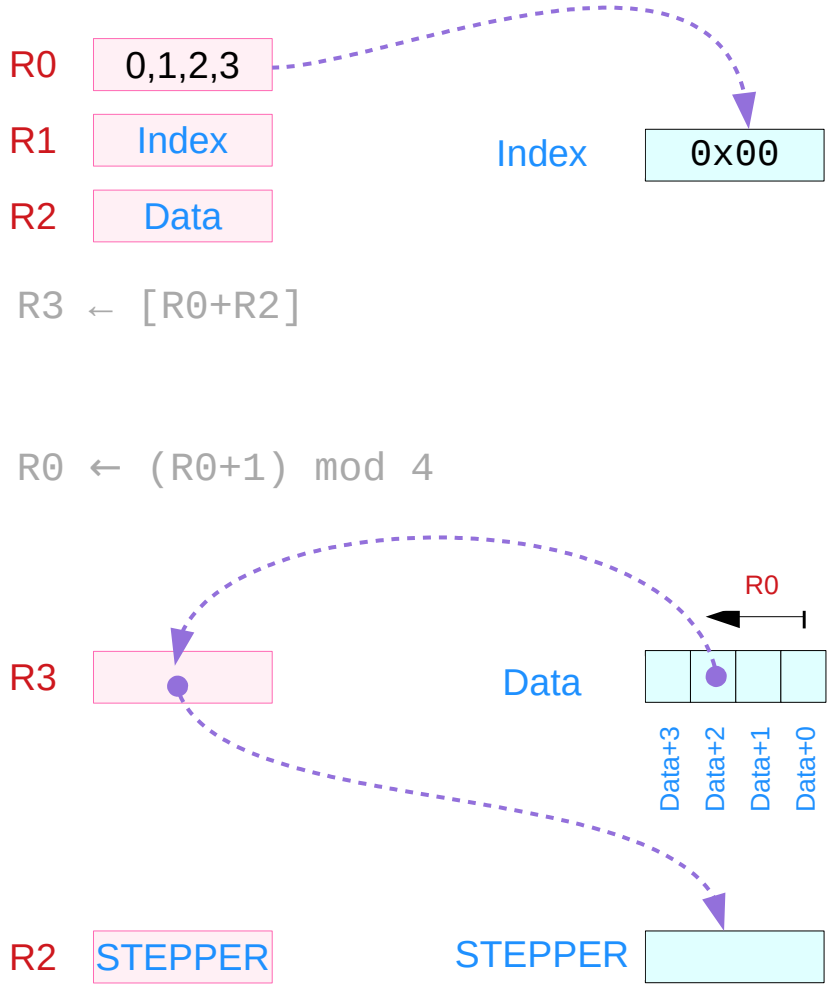
Stepper example (1)

```
                AREA    DATA, ALIGN=2
Index           SPACE   4

                AREA    |.text|, CODE, READONLY, ALIGN=2
                THUMB
Data            DCB     0x05, 0x06, 0x0A, 0x09
STEPPER         EQU     0x4000703C

Stepper_Init
                PUSH    {R4, LR}
                BL      GPIO_Init
R1  STEPPER     LDR     R1, =STEPPER
                MOV     R0, #0x09
                STR     R0, [R1]
                MOV     R0, #0
                LDR     R1, =Index
                STR     R0, [R1]
                POP     {R4, PC}
```



Stepper example (2)

```
; Move one step clockwise
Stepper_CW
            LDR     R1, =Index
            LDR     #0, [R1]
            LDR     R2, =Data
            LDRB    R3, [R0, R2]    ; R3 ← [R0+R2]
            LDR     R2, =STTEPPER
            STR     R3, [R2]
            ADD     R0, R0, #1      ; R0 ← (R0+1) mod 4
            AND     R0, R0, #3
            STR     R0, [R1]
            BX      LR
```



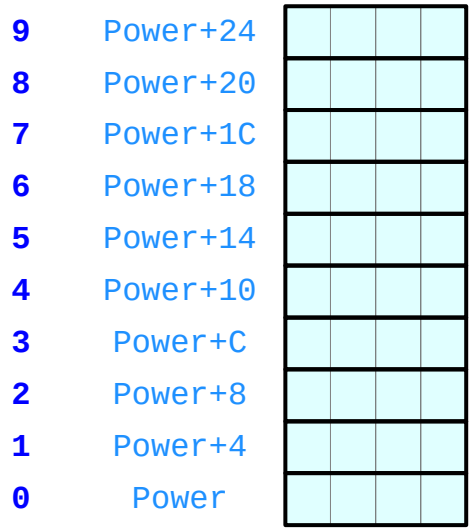
Stepper example (3)

```
#define     STEPPER  (* ((volatile uint32_t *) 0x4000703C

const unit8_t Data[4] = {0x05, 0x06, 0x0A, 0x09};

unit32_t static Index;

Void Stepper_Init(void) {
    GPIO_Init();
    STEPPER = 0x09;
    Index = 0;
}

Void Stepper_CW(void) {
    STEPPER = Data[Index];
    Index = (Index+1) & 0x03;
}
```

Powers example (1)

```
            AREA    |.text|, CODE, READONLY, ALIGN=2
Power       DCD     1, 10, 100, 1000, 10000, 100000
            DCD     1000000, 10000000, 100000000, 1000000000


; Input : R0 = x        Output: R0=10^x
Power       LSL R0, R0, #2      ; multiply by 4 to get byte address
            LDR R1, =Power
            LDR R0, [R0, R1]
            BX LR
```

```
Const uint32_t Powers[10]
 = {1, 10, 100, 1000, 10000, 100000
    1000000, 10000000, 100000000, 1000000000}

unit32_t power(uint32_t x) {
    return Powers[x];
}
```

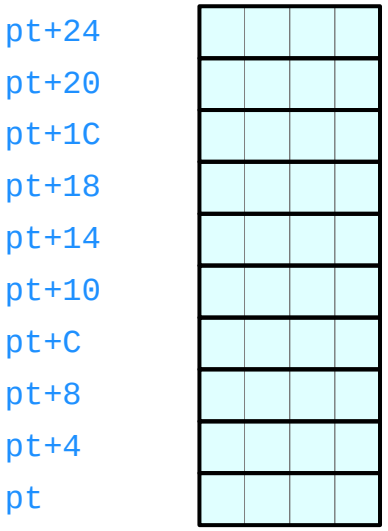


Powers example (2)

```
Const uint32_t Powers[11]
 = {10, 1, 10, 100, 1000, 10000, 100000
    1000000, 10000000, 100000000, 1000000000}

uint32_t power(unit32_t x) {
    if (x<Powers[0])            ; x < 10
        return Powers[x+1];
    return 0xFFFFFFFF;          ; x >= 10 out of bounds
}
; x     =   0, 1, 2, 3, ..., 9
; x+1   =   1, 2, 3, 4, ... ,10
```

pt++;

pt = pt+1;



C pointer address **assembly address**

Introduction to ARM Cortex-M Microcontrollers – Embedded Systems, Jonathan W. Valvano

Constant data examples

```
const uint16_t Data1[5]  = {0, 3, 4, 1, 65535};
const uint16_t Data2[10] = {1,2,3,4,5,6,7,8,9,65535};
const uint16_t Data3[1]  = {65535};
const uint16_t Data4[5]  = {0, 3, -4, 1, -32768};
const uint16_t Data5[10] = {1,-2,3,4,-5,6,-7,8,-9,-32768};
const uint16_t Data6[1]  = {-32768};
```

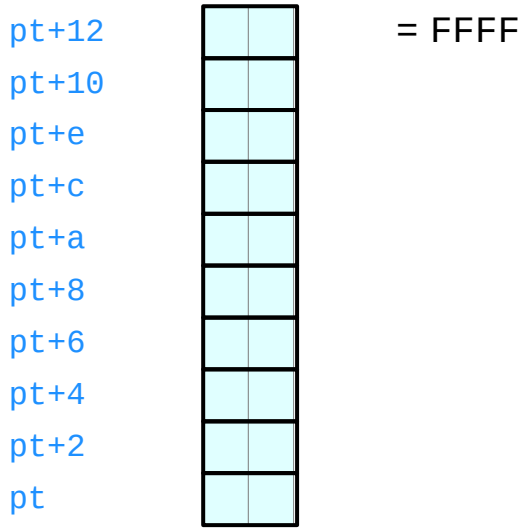
Summing elements of an array – **unsigned int** (1)

```
uint16_t Sum(uint16_t *pt) {
  uint16_t result=0;
  while (*pt != 65535) {
    result += (*pt);
    pt++;
  }
  return result;
}
```

C pointer address **assembly address**

pt++;

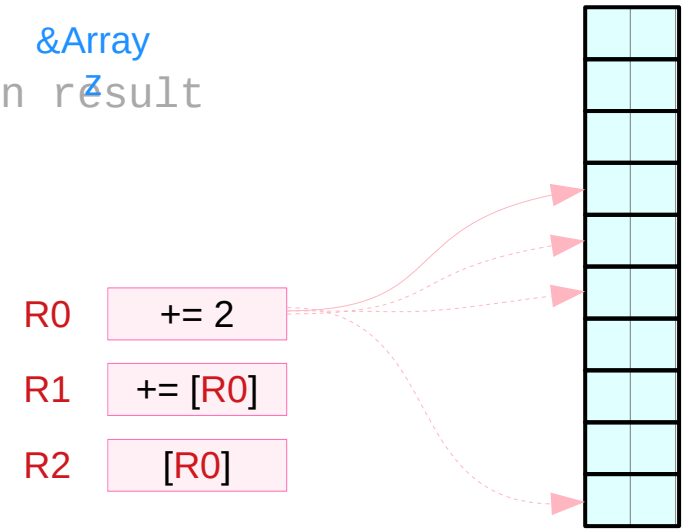
pt = pt+1;



Introduction to ARM Cortex-M Microcontrollers – Embedded Systems, Jonathan W. Valvano

Summing elements of an array – **unsigned int** (2)

```
; input:   R0 = &Array     ; address
; output:  R0 = Sum        ; sum value
Sum     MOV     R1, #0
        MOV     R3, #65535
loop    LDRH    R2, [R0]    ; value from array
        CMP     R2, R3      ; termination?
        BEQ     done
        ADD     R1, R1, R2  ; accumulate
        ADD     R0, #2      ; next halfword
        B       loop
done    MOV     R0, R1      ; return result
        BX      LR
```



Introduction to ARM Cortex-M Microcontrollers – Embedded Systems, Jonathan W. Valvano

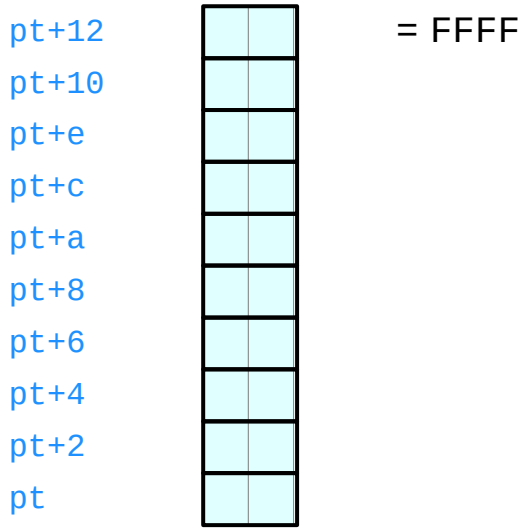
Summing elements of an array – **signed int** (1)

```
uint16_t Sum(int16_t *pt) {
  uint16_t result=0;
  while (*pt != -32768) {
    result += (*pt);
    pt++;
  }
  return result;
}
```

C pointer address **assembly address**

pt++;

pt = pt+1;

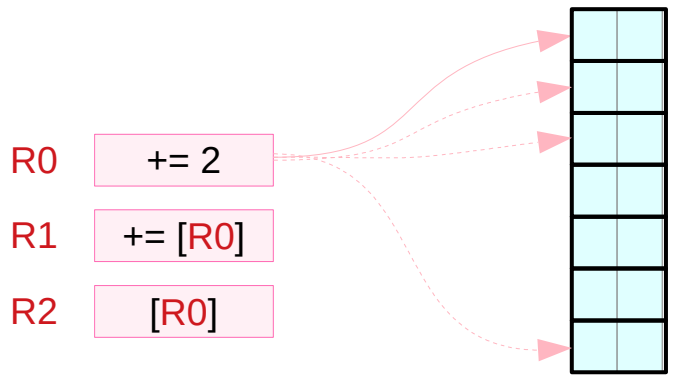


Introduction to ARM Cortex-M Microcontrollers – Embedded Systems, Jonathan W. Valvano

Summing elements of an array – **signed int** (2)

```
; input:   R0 = &Array
; output:  R0 = Sum
Sum     MOV     R1, #0
loop    LDRSH   R2, [R0]     ; value from array
        CMP     R2, #-32768  ; termination?
        BEQ     done
        ADD     R1, R1, R2
        ADD     R0, #2       ; next halfword
        B       loop
done    MOV     R0, R1       ; return result
        BX      LR
```

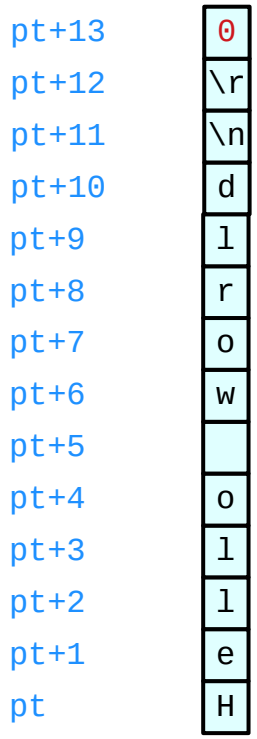
```
; input:   R0 = &Array
; output:  R0 = Sum
Sum     MOV     R1, #0
        MOV     R3, #65535
loop    LDRH    R2, [R0]
        CMP     R2, R3
        BEQ     done
        ADD     R1, R1, R2
        ADD     R0, #2
        B       loop
done    MOV     R0, R1
        BX      LR
```



Introduction to ARM Cortex-M Microcontrollers – Embedded Systems, Jonathan W. Valvano

String Constants

```
const char Hello[] = “Hello world\n\r”;

Hello   DCB     ”Hello world\n\r”, 0      ; 0 : null termination

OutString(Hello);
OutString(&Hello[0]);
OutString(“Hello world\n\r”);
```

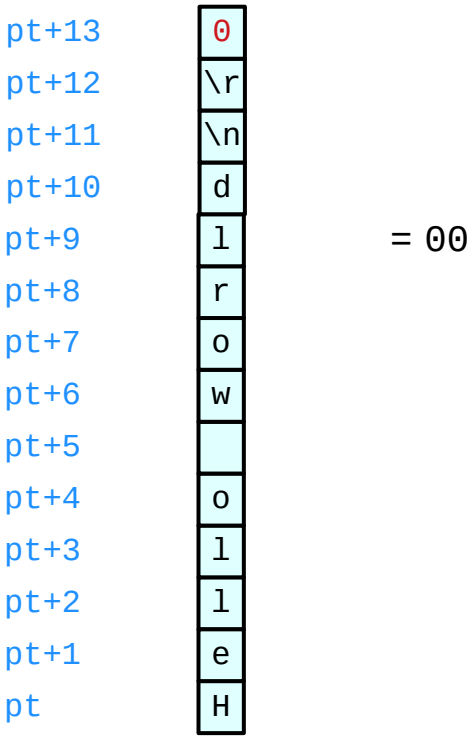


OutString

```
// display a string
void OutString(char *pt) {
  while (*pt) {    ; 0 (null termination)
     outChar(*pt); // output
     pt++;
  }
}
```

pt++;

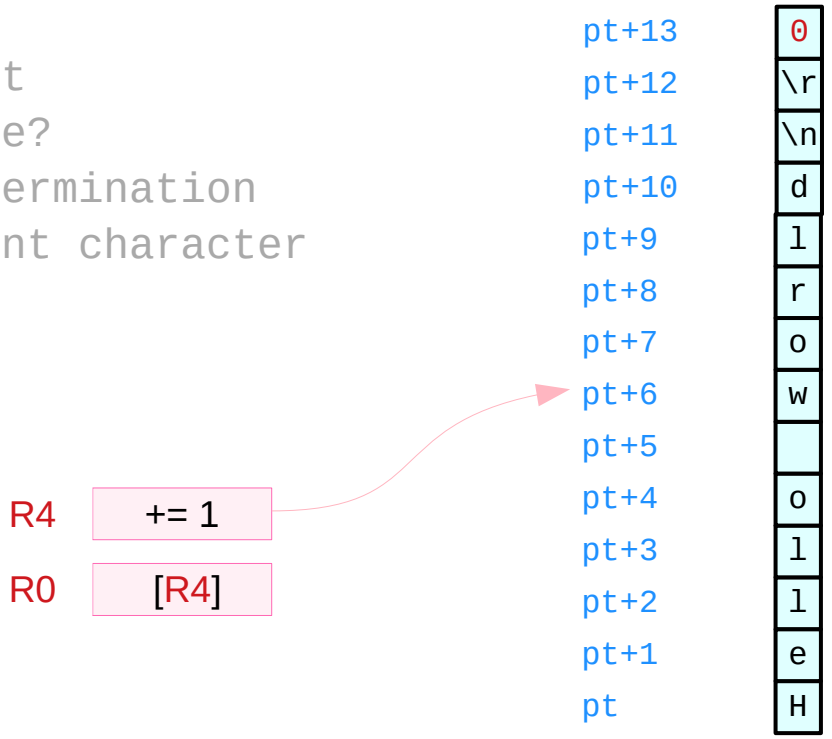
pt = pt+1;



Introduction to ARM Cortex-M Microcontrollers – Embedded Systems, Jonathan W. Valvano

OutString

```
; input: R0 points to
; string
OutString
        PUSH    {R4, LR}
        MOV     R4, R0
loop    LDRB    R0, [R4]
        ADD     R4, #1          ; next
        CMP     R0, #0          ; done?
        BEQ     done            ; 0 termination
        BL      OutChar         ; print character
        B       loop
done    POP     {R4, LR}
```



Introduction to ARM Cortex-M Microcontrollers – Embedded Systems, Jonathan W. Valvano

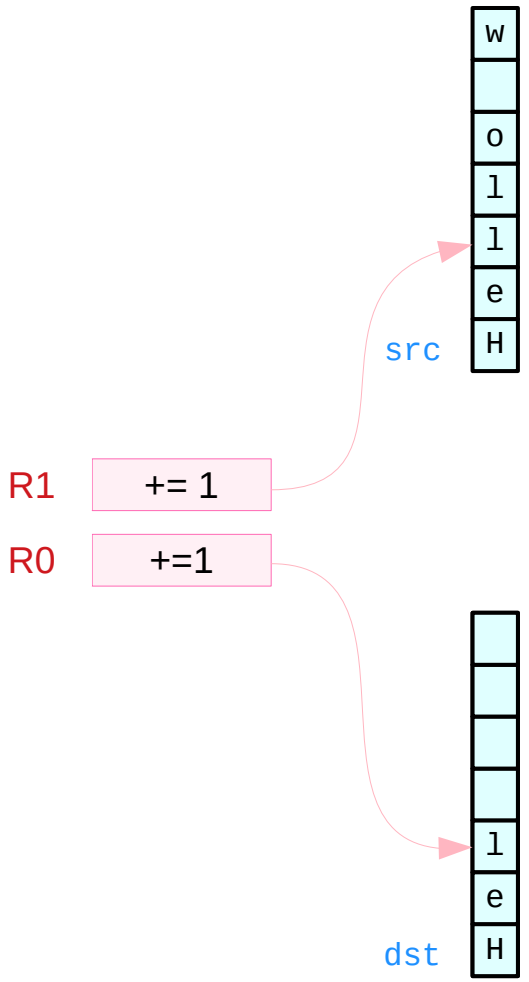
String Copy

```
char String1[20];
char String2[20];

   String1 = “Hello”;   /**** illegal: not lvalue ****/
   String2 = String1;   /**** illegal: not lvalue ****/

   strcpy(String1, “Hello”);
   strcpy(String2, String1);
```

strcpy

```
; Input, R0=&dst[0] R1=&src[0]
strcpy  LDRB    R2, [R1]    ; source data
        STRB    R2, [R0]    ; copy
        CMP     R2, #0      ; termination?
        BEQ     done
        ADD     R1, #1      ; next
        ADD     R0, #1
        B       strcpy
done    BX      LR
```

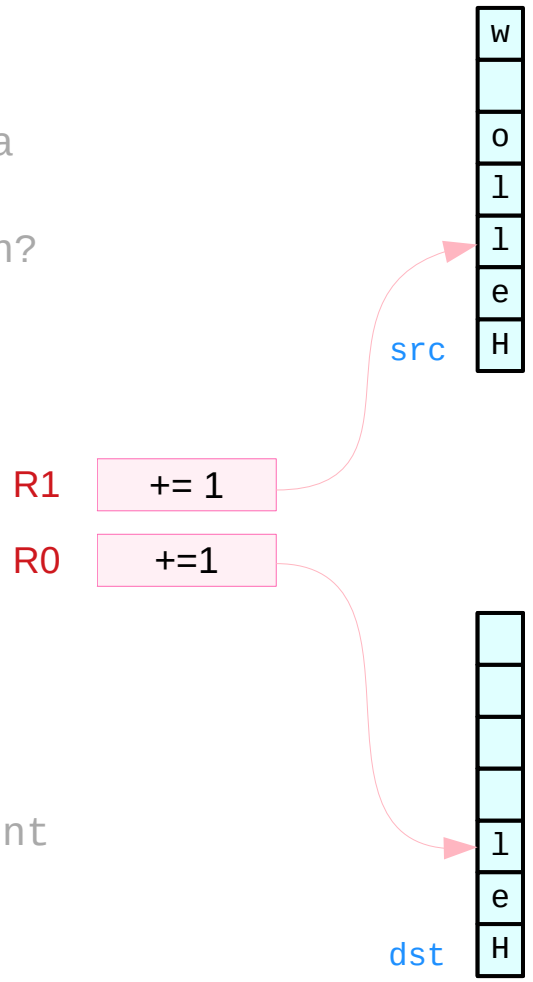
```
// copy string form source to dest
void strcpy(char *dst, char *src) {
  While (*src) {
    *dst = *src;  // copy
    dst++;        // next
    src++;
  }
  *dst = *src;    // termination
}
```



Introduction to ARM Cortex-M Microcontrollers – Embedded Systems, Jonathan W. Valvano

strcpy using post-indexing – faster

```
; Input, R0=&dst[0] R1=&src[0]
; faster version

strcpy  LDRB    R2, [R1], #1    ; source data
        STRB    R2, [R0], #1    ; copy
        CMP     R2, #0          ; termination?
        BEQ     done
        B       strcpy
done    BX      LR
```




```
// another version
void strcpy(char *dst, char *src) {
char data;
  do {
    data = (dst++ = *src++); // post-increment
  } while(data);
}
```

Structures

```
struct player {
  uint8_t   Xpos;
  uint8_t   Ypos;
  uint16_t  LifePoints;
};

typdef struct player playerType;

playerType Sprite;
    Sprite.Xpos = 10;
    Sprite.Ypos = 20;
    Sprite.LifePoints = 12000;

playerType *pter;
    Ptr = &Sprite;
    Ptr->Xpos = 10;
    Ptr->Ypos = 2
    Ptr->LifePoints = 12000;
```

Structures

```
// move to center and
// add life

void MoveCenter(playerType *pt) {
  pt->Xpos = 50;
  pt->Ypos = 50;
  if (pt->LifePoints < 65535) {
    pt->LifePoints++;
  }
}
```

C Operator Precedence

```
Operator Precedence

++ --               Suffix/postfix increment and decrement  Left-to-right
()                  Function call
[]                  Array subscripting
.                   Structure and union member access
->                  Structure and union member access through pointer
(type){list}        ompound literal(C99)
```

```
(pt->LifePoints)++   ⇨   ++ is considered as having a higher precedence (O)
pt->(LifePoints++)   ⇨   -> is considered as having a higher precedence (X)
```

Structure example

```
; input: R0 = &PlayerType
Xpos        EQU     0
Ypos        EQU     1
Lpnt        EQU     2

MoveCenter
    MOV     R1, #50
    STRB    R1, [R0, #Xpos]     ; [R0, #0]      R0+0
    STRB    R1, [R0, #Ypos]     ; [R0, #1]      R0+1
    LDRH    R1, [R0, #Lpnt]     ; [R0, #2]      R0+2
    LDR     R2, =65535
    CMP     R1, R2              ; st max?
    BHS     skip
    ADD     R1, #1              ; more life
    STRH    R1, [R0, #Lpnt]
skip
    BX      LR
```

Condition Codes

HS,	CS	>=	Unsigned	GE	>=	Signed
Lo,	CC	<	Unsigned	LT	<	Signed
HI		>	Unsigned	GT	>	Signed
LS		<=	Unsigned	LE	<=	Signed

```
H : HIgher
L : LOwer
S : Same

CS : Carry Set
CC : Carry Clear
```

Clearing an array

```
clear1(int array[], int size)
{
    int i;
    for (i=0; i<size; i+=1)
        array[i] = 0;
}


clear2(int array[], int size)
{
    int *p;
    for (p=array; p<array+size; p+=1)
        *p = 0;
}
```

Clearing an array – method 1

```
array   RN  0   ; 1st argument address of array      ; R0
n       RN  1   ; 2nd argument size of array         ; R1
i       RN  2   ; local variable i                   ; R2
zero    RN  3   ; temporary to hold constant 0       ; R3
size    RN  6                                        ; R6
```

```
    MOV     i,  0
    MOV     zero, 0
loop1:
    STR     zero, [array, i, LSL#2]
    ; array + 4*i : pre-indexing
    ADD     i, i, #1
    CMP     i, R6
    BLT     loop1
```

```
    MOV     R2, 0
    MOV     R3, 0
loop1:
    STR     R3, [R0, R2, LSL#2]
    ; array + 4*i : pre-indexing
    ADD     R2, R2, #1
    CMP     R2, R6
    BLT     loop1
```

Computer Organization and Design ARM Edition: The Hardware Software Interface By David A. Patterson, John L. Hennessy

Clearing an array – method 2

```
array   RN  0   ; 1st argument address of array      ; R0
n       RN  1   ; 2nd argument size of array         ; R1
p       RN  2   ; local variable p                   ; R2
zero    RN  3   ; temporary to hold constant 0       ; R3
asize   RN  12  ; address of array[size]             ; R12
size    RN  6                                        ; R6
```

```
    MOV     p,  array
    MOV     zero, #0
loop2:
    STR     zero, [p], #4
            ; post-indexing
    ADD     asize, array, size, LSL #2
            ; asize = array + size*4
    CMP     p, asize
    BLT     loop2
```

```
    MOV     R2, R0
    MOV     R3, #0
loop2:
    STR     R3, [R2], #4
            ; post-indexing
    ADD     R12, R0, R6, LSL #2
            ; asize= array+size*4
    CMP     R2, R12
    BLT     loop2
```

Computer Organization and Design ARM Edition: The Hardware Software Interface By David A. Patterson, John L. Hennessy

Clearing an array – method 3

```
array   RN  0   ; 1st argument address of array      ; R0
n       RN  1   ; 2nd argument size of array         ; R1
p       RN  2   ; local variable p                   ; R2
zero    RN  3   ; temporary to hold constant 0       ; R3
asize   RN  12  ; address of array[size]             ; R12
size    RN  6                                        ; R6
```

```
    MOV     p,  array
    MOV     zero, #0
    ADD     asize, array, size, LSL #2
            ; asize = array + size*4
loop2:
    STR     zero, [p], #4
            ; post-indexing
    CMP     p, asize
    BLT     loop2
```

```
    MOV     R2, R0
    MOV     R3, #0
    ADD     R12, R0, R6, LSL #2
            ; asize= array+size*4
loop2:
    STR     R3, [R2], #4
            ; post-indexing
    CMP     R2, R12
    BLT     loop2
```

Computer Organization and Design ARM Edition: The Hardware Software Interface By David A. Patterson, John L. Hennessy

No indexing example

```
for (i = 0; i < 100; i++)
  a[i] = i;

.text

.global main
main:
    ldr r1, addr_a          ; r1 ← &a
    mov r2, #0              ; r2 ← 0
loop:
    cmp r2, #100            ; reached 100?
    beq end                 ; If so, leave the loop
    add r3, r1, r2, LSL #2  ; r3 ← r1 + (r2*4)
    str r2, [r3]            ; *r3 ← r2
    add r2, r2, #1          ; r2 ← r2 + 1
    b loop                  ; Go to the loop
end:
    bx lr

addr_a: .word a
```

https://thinkingeek.com/2013/01/27/arm-assembler-raspberry-pi-chapter-8/

No indexing example

Pre-indexing without updating

```
[Rs, +#imm]
[Rs, -#imm]

[Rs, +Rn]
[Rs, -Rn]

[Rs, +Rn, shift #imm]
[Rs, -Rn, shift #imm]
```

Post-indexing always updating

```
[Rs], #+imm
[Rs], #-imm

[Rs], +Rn
[Rs], -Rn

[Rs], +Rn, shift #imm
[Rs], -Rn, shift #imm
```

Post-indexing with updating

```
[Rs, +#imm]!
[Rs, -#imm]!

[Rs, +Rn]!
[Rs, -Rn]!

[Rs, +Rn, shift #imm]!
[Rs, -Rn, shift #imm]!
```

https://thinkingeek.com/2013/01/27/arm-assembler-raspberry-pi-chapter-8/

Pre-indexing without updating

```
mov r2, #3              ; r2 ← 3
str r2, [r1, #+12]      ; *(r1 + 12) ← r2


mov r2, #3              ; r2 ← 3
mov r3, #12             ; r3 ← 12
str r2, [r1, +r3]       ; *(r1 + r3) ← r2


mov r2, #3              ; r2 ← 3
add r3, r1, r2, LSL #2  ; r3 ← r1 + r2*4
str r2, [r3]            ; *r3 ← r2
```

https://thinkingeek.com/2013/01/27/arm-assembler-raspberry-pi-chapter-8/

Indexing with updating

```
add r3, r1, r2, LSL #2      ; r3 ← r1 + r2*4
str r2, [r3]                ; *r3 ← r2

str r2, [r1]                ; *r1 ← r2
add r1, r1, #4              ; r1 ← r1 + 4

str r2, [r1], #4            ; *r1 ← r2 and then r1 ← r1 + 4



/* Wrong syntax */
str r2, [r1] "and then" add r1, r1, #4

/* Wrong syntax */
str r2, [add r1, r1, #4]
```

https://thinkingeek.com/2013/01/27/arm-assembler-raspberry-pi-chapter-8/

Post-indexing always updating

```
loop:
    cmp r2, #100            ; reached 100?
    beq end                 ; If so, leave the loop
    str r2, [r1], #4        ; *r1 ← r2 then r1 ← r1 + 4
    add r2, r2, #1          ; r2 ← r2 + 1
    b loop                  ; Go to the loop
end:
```

https://thinkingeek.com/2013/01/27/arm-assembler-raspberry-pi-chapter-8/

Pre-indexing with updating

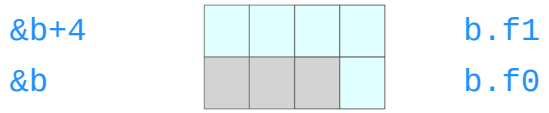
```
ldr r2, [r1, #+12]!     ; r1 ← r1 + 12 then r2 ← *r1
add r2, r2, r2          ; r2 ← r2 + r2
str r2, [r1]            ; *r1 ← r2 where r1 is updated
```

```
a[3] = a[3] + a[3] or
a[3] *= 2
```

https://thinkingeek.com/2013/01/27/arm-assembler-raspberry-pi-chapter-8/

Structure

```
struct my_struct
{
  char f0;
  int f1;
} b;

b.f1 = b.f1 + 7;
```




```
ldr r2, [r1, #+4]!      ; r1 ← r1 + 4 then r2 ← *r1
add r2, r2, #7          ; r2 ← r2 + 7
str r2, [r1]            ; *r1 ← r2
```

https://thinkingeek.com/2013/01/27/arm-assembler-raspberry-pi-chapter-8/

Bubble Sort – C

```
// C program for implementation of Bubble sort – www.geeksforgeeks.org
#include <stdio.h>

void swap(int *xp, int *yp)
{
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}

void bsort(int a[], int n)
{
   int i, j;
   for (i = 0; i < n-1; i++)

       // Last i elements are already in place
       for (j = 0; j < n-i-1; j++)
           if (a[j] > a[j+1])
              swap(&a[j], &a[j+1]);
}
```

https://www.geeksforgeeks.org/bubble-sort/

Bubble Sort – Optimized Pseudocode

```
procedure bubbleSort(A : list of sortable items)
    n := length(A)
    repeat
        swapped := false
        for i := 1 to n - 1 inclusive do
            if A[i - 1] > A[i] then
                swap(A[i - 1], A[i])
                swapped = true
            end if
        end for
        n := n - 1
    until not swapped
end procedure
```

https://www.geeksforgeeks.org/bubble-sort/

Bubble Sort Assembly Code (1)

```
/*
    Copyright 2019, Andrew C. Young <andrew@vaelen.org>

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <https://www.gnu.org/licenses/>.
*/
```

https://vaelen.org/post/arm-assembly-sorting/

Bubble Sort Assembly Code (2)

```
// Bubble Sort

// Exported Methods
    .global bsort

// Code Section

bsort:
    // Bubble sort an array of 32bit integers in place
    // Arguments: R0 = Array location, R1 = Array size
    PUSH    {R0-R6,LR}          // Push registers on to the stack
```

https://vaelen.org/post/arm-assembly-sorting/

Bubble Sort Assembly Code (3)

```
bsort_next:                         // Check for a sorted array
    MOV     R2,#0                   // R2 = i Current Element Number
    MOV     R6,#0                   // R6 = Number of swaps
bsort_loop:                         // Start loop
    ADD     R3,R2,#1                // R3 = i+1 Next Element Number
    CMP     R3,R1                   // Check for the end of the array
    BGE     bsort_check             // When we reach the end, check for changes
    LDR     R4,[R0,R2,LSL #2]       // R4 = a[i] Current Element Value
    LDR     R5,[R0,R3,LSL #2]       // R5 = a[i+1] Next Element Value
    CMP     R4,R5                   // Compare element values
    STRGT   R5,[R0,R2,LSL #2]       // If R4 > R5, store current value at next
    STRGT   R4,[R0,R3,LSL #2]       // If R4 > R5, Store next value at current
    ADDGT   R6,R6,#1                // If R4 > R5, Increment swap counter
    MOV     R2,R3                   // Advance to the next element
    B       bsort_loop              // End loop
```

https://vaelen.org/post/arm-assembly-sorting/

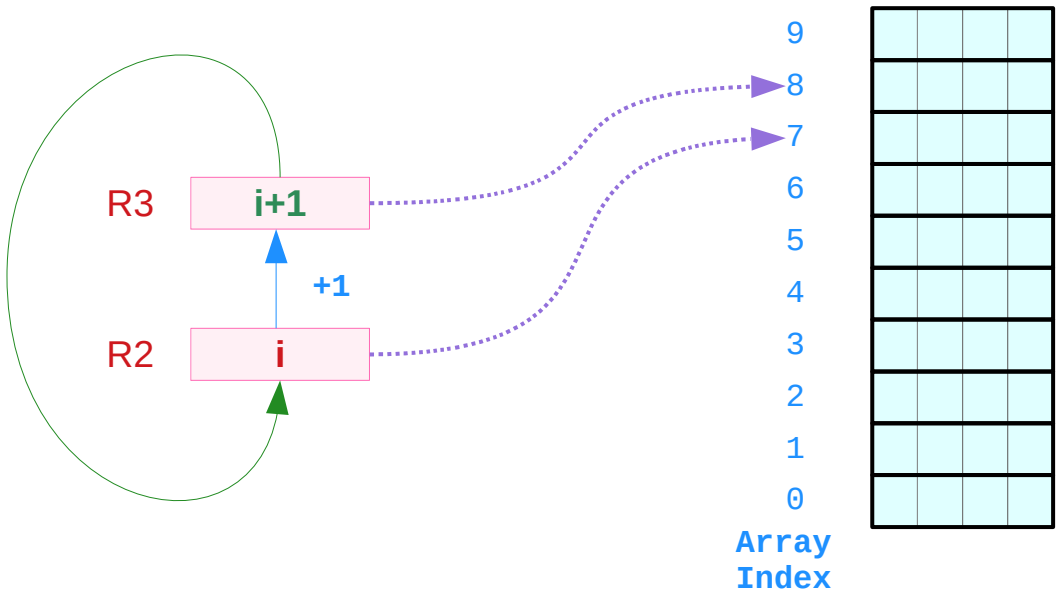
Bubble Sort Assembly Code (4)

```
bsort_check:                    // Check for changes
    CMP     R6,#0               // Were there changes this iteration?
    SUBGT   R1,R1,#1            // Optimization: skip last value in next loop
    BGT     bsort_next          // If there were changes, do it again
bsort_done:                     // Return
    POP     {R0-R6,PC}          // Pop the registers off of the stack
```

https://vaelen.org/post/arm-assembly-sorting/

Bubble Sort Assembly Code – updating index

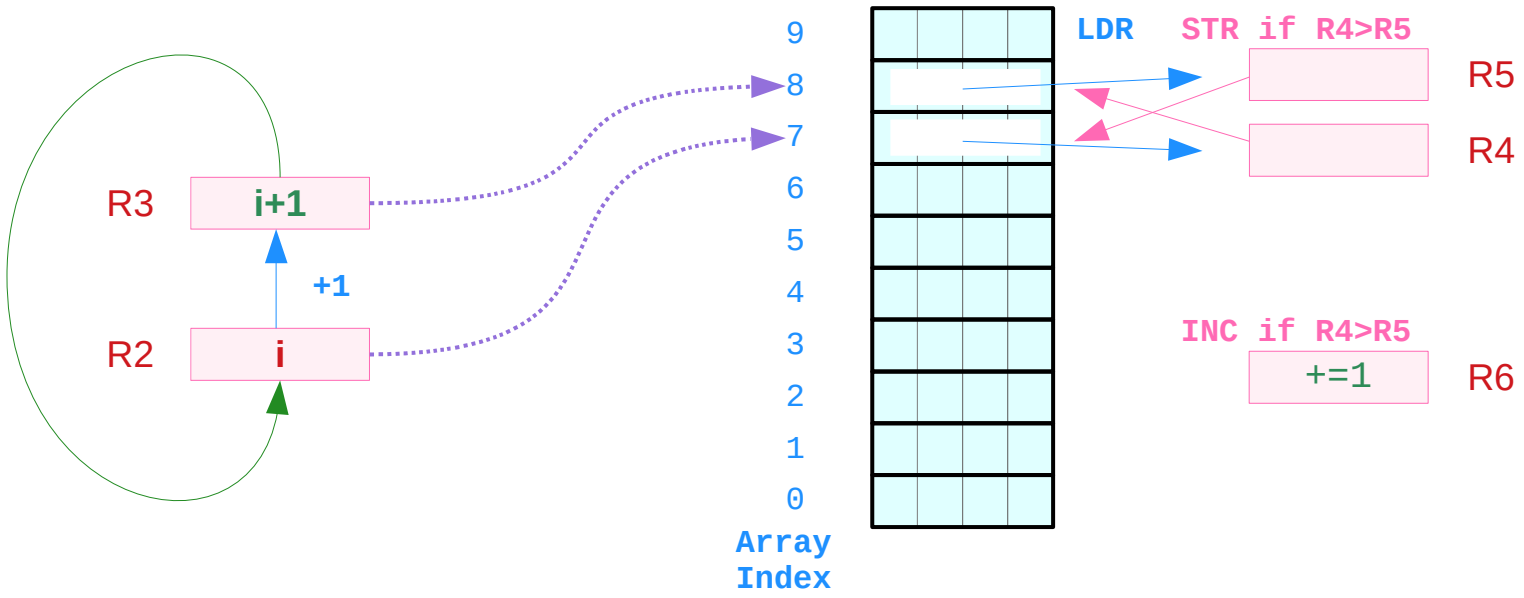
```
MOV     R2,#0           // R2 = i Current Element Number

ADD     R3,R2,#1        // R3 = i+1 Next Element Number

MOV     R2,R3           // Advance to the next element
```



https://vaelen.org/post/arm-assembly-sorting/

Bubble Sort Assembly Code – swapping operation

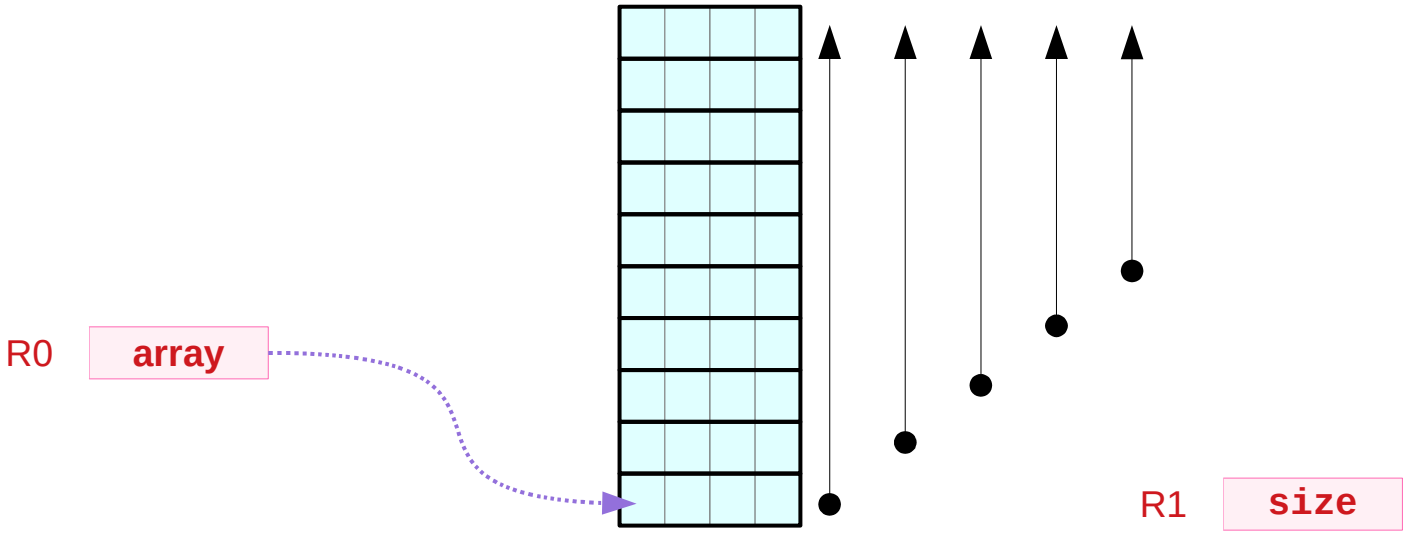
```
LDR     R4,[R0,R2,LSL #2]   // R4 = a[i] Current Element Value
LDR     R5,[R0,R3,LSL #2]   // R5 = a[i+1] Next Element Value
CMP     R4,R5               // Compare element values
STRGT   R5,[R0,R2,LSL #2]   // If R4 > R5, store current value at next
STRGT   R4,[R0,R3,LSL #2]   // If R4 > R5, Store next value at current
ADDGT   R6,R6,#1            // If R4 > R5, Increment swap counter
```



https://vaelen.org/post/arm-assembly-sorting/

Bubble Sort Assembly Code – loop condition

```
    CMP     R3,R1               // Check for the end of the array
    BGE     bsort_check         // When we reach the end, check for changes

bsort_check:                    // Check for changes
    CMP     R6,#0               // Were there changes this iteration?
    SUBGT   R1,R1,#1            // Optimization: skip last value in next loop
    BGT     bsort_next          // If there were changes, do it again
bsort_done:                     // Return
    POP     {R0-R6,PC}          // Pop the registers off of the stack
```



https://vaelen.org/post/arm-assembly-sorting/

Pointer access to an array

References

[1] ftp://ftp.geoinfo.tuwien.ac.at/navratil/HaskellTutorial.pdf
[2] https://www.umiacs.umd.edu/~hal/docs/daume02yaht.pdf