ARMv8 Assembly Programming

Humayun Kabir
Professor, CS, Vancouver Island University, BC, Canada

ARMv8 Assembly Programming: Outline

- Basics
- Selection
- Iteration
- Calling Functions
- Writing Functions
- Aggregate Data Types

ARMv8 Assembly Programming: Basics

- **Global** or **static variables** are placed in the **data section** of a program.
- A **data section** is declared using **.data** assembler directive.
- A data section continues until a new section starts.

```
                .data           //data section starts
number1:        .word 100       //number1 = 100
                .skip 4, 0      //skip 4 bytes to align number2 at multiple of 8 address
number2:        .word 200       //number2 = 200
                .align 3, 0     //align number3 at multiple of 8 address
number3:        .word 300       //number3 = 300
```

ARMv8 Assembly Programming: Basics

- **Code** or **assembly instructions** are placed in the **text section** of a program.
- A **text section** is declared using **.text** assembler directive.
- A text section continues until a new section starts.
- There must be a **main** function in a program, and it should be made **global** for the linker.
- Assembler directive .global or .globl is used to make a symbol global for the linked.

ARMv8 Assembly Programming: Basics

```
                .text                   //text section starts
                .global main            //main is made global
main:           ldr x9, =number1        //main function starts
                ldr x9, [x9]
                ldr x10, =number2
                ldr x10, [x10]
                ldr x11, =number3
                ldr x11, [x11]          //Codes for main function
                add x0, x9, x10
                add x0, x0, x11
                ret lr
```

ARMv8 Assembly Programming: Basics

- Function parameters are passed and received through x0 to x7 registers.
- Function results are returned and received through x0 to x7 registers.
- Lower numbered register should not be skipped to use a higher numbered register for parameters and for returns.
- Local variables of a function is programmed using registers, preferably using x9 to x15. Registers x0 to x7 can also be used to program local variables.

ARMv8 Assembly Programming: Basics

- A function is called using bl *fname* instruction. It automatically copies the return address into link register (lr or x30).
- A function is returned using ret lr instruction. Mentioning lr in ret instruction is optional.
- If a function calls another function current lr value must be **saved** onto stack and it should be **retrieved** from stack into lr before **returning** from the function.
- A function needs to use stack to save the return address (lr), frame pointer (fp), local variables, and to pass parameters to its called functions.
- The portion of the stack used by a function is called stack frame.

ARMv8 Assembly Programming: Basics

- The **start** of the **current stack frame** is pointed by the frame pointer (fp or x29).
- The **stack frame** of a function is removed or wrapped up when the **function returns**, and the frame pointer **points** back to the **start** of the **previous stack frame**.
- The **successive calls** of **many functions** builds many stack frames on the stack one after another and the **boundaries** of the **stack frames** are **linked together** with a linked list and the frame pointer effectively points to the **top** of this **linked list.**
- The **successive returns** of many functions **rewinds** the stack frames following through the linked list whose **top** is always **pointed** by the frame pointer.

ARMv8 Assembly Programming: Basics

```
1
2               .data                           //Start of data segment
3   greet:      .asciz "Hello World!\n"         //Null terminated string inside the data segment
4
5               .text                           //Start of text or code segment
6               .global helloworld              //Making 'main' symbol global for the linker
7   helloworld:                                 //Start of the 'main' function
8               stp fp, lr, [sp, #-16]!         //Expand the stack and save the current return address
9                                               //and the frame poniter on the stack
10              mov fp, sp                      //Make the frame pointer pointing the start of the
11                                              //current frame
12              ldr x0, =greet                  //Load the address of the 'greet' string to x0
13              bl printf                       //Call 'printf' function passing the address of the
14                                              //string through x0
15              mov x0, xzr                     //Set zero to x0 in order to return zero from 'main'
16
17              mov sp, fp                      //Rewind the stack
18              ldp fp, lr, [sp], #16           //Retrieve the saved return address and the frame pointer
19                                              //and shrink the stack
20              ret lr                          //Return from the 'main' function
21
```

[label:] [assembler directive or assembly instruction] // comments

Selection C Code

```
static int number1 = 10;
static int number2 = 4;

int selection()  {
        if ( number1 < number2 ) {
                return number1;
        }
        else {
                return number2;
        }
}
```

Selection ARMv8 Assembly Code: Selection

```
1               .data
2 number1:      .word 10          // static int number1=10;
3               .skip 4, 0
4 number2:      .word 4           // static int number2=4;
5               .skip 4, 0
6
7           .text
8           .global selection1
9
10 selection1:
11          ldr x9, =number1      // load address of number1 into x9
12          ldr x9,[x9]           // load value of number1 into x9
13
14          ldr x10, =number2     // load address of number2 into x10
15          ldr x10,[x10]         // load value of number2 into x10
16
17          cmp x9, x10           // compare number1 (x9) with number2 (x10)
18 if:      blt then              // if number1 >= number2 branch to else
19 else:
20          mov x0, x10           // assign number1 to result
21          b endif               // branch to endif in order to skip the exceution of else block
22 then:
23          mov x0, x9            // else assign number2 to result
24 endif:
25          ret lr
```

Selection ARMv8 Assembly Code: Selection

```
1               .data
2   number1:    .word 10          // static int number1=10;
3               .skip 4, 0
4   number2:    .word 4           // static int number2=4;
5               .skip 4, 0
6
7
8           .text
9           .global selection2
10
11  selection2:
12          ldr x9, =number1      // load address of number1 into x9
13          ldr x9,[x9]           // load value of number1 into x9
14
15          ldr x10, =number2     // load address of number2 into x10
16          ldr x10,[x10]         // load value of number2 into x10
17
18          cmp x9, x10           // compare number1 (x9) with number2 (x10)
19          bge else              // if number1 >= number2 branch to else
20  then:
21          mov x0, x9            // assign number1 to result
22          b endif               // branch to endif in order to skip the exceution of else block
23  else:
24          mov x0, x10           // else assign number2 to result
25  endif:
26          ret lr
```

Iteration C Code

```
int iteration() {
        int sum = 0;
        int i;
        for(i=0; i<10; i++) {sum += i;}
        return sum;
}
```

```
int iteration() {
        int sum = 0;
        int i= 0;
        while(i<10) { sum += i;  i++; }
        return sum;
}
```

```
int iteration() {
        int sum = 0;
        int i= 0;
        do{ sum += i;  i++; } while(i<10);
        return sum;
}
```

ARMv8 Assembly Code: Iteration

```
1
2               .text
3               .global iteration1
4   iteration1:
5               mov x9, xzr             // Use x9 for i; i=0
6               mov x10, xzr            // Use x10 as sum and initialize sum = 0
7
8   while:      cmp x9, #10             // Compare i with 10
9               bge endwhile            // end loop if i >= 10
10              add x10, x10, x9        // sum += i
11              add x9, x9, #1          // i++
12              b while                 // repeat while loop test
13  endwhile:
14              mov x0, x10             //Return sum through x0
15              ret lr                  //Return from the main function
```

ARMv8 Assembly Code: Iteration

```
1
2               .text
3               .global iteration2
4   iteration2:
5               mov x9, xzr             // Use x9 for i; i=0
6               mov x10, xzr            // Use x10 as sum and initialize sum = 0
7
8   do:         add x10, x10, x9        // sum += i
9               add x9, x9, #1          // i++
10              cmp x9, #10             // Compare i with 10
11              blt do                  // end loop if i >= 10
12  enddo:
13              mov x0, x10             //Return sum through x0
14              ret lr                  //Return from the main function
15
```

ARMv8 Rules for Calling a Function

- Caller function needs to **expand** the **stack pointer** and **push** the **link register (lr)** and **frame pointer (fp)** current values onto the **stack**.
- **Stack address grows downward,** and it must be **minimum 16 scaled**, i.e., stack pointer should be **subtracted** at **least 16** to expand.
- Before return caller function needs to **pop** the saved **link register** and **frame pointer** values from the **stack** into ***lr*** and **fp** registers respectively and shrink the stack pointer accordingly.
- **Stack address shrinks upward**, i.e., should be added at least **16**.

ARMv8 Calling C Library Functions

```
1
2               .data
3   prompt:     .asciz "\nEnter a number: "
4               .align 8
5   frmt_int:   .asciz "%d"
6               .align 8
7   msg_label:  .asciz "You entered %d\n"
8   number:     .long 0
9
10          .text
11          .global callfunction
12  callfunction:
13          stp fp, lr, [sp, #-16]! // expand the stack 16 bytes and
14                                  // push link register and frame pointer onto stack
15          mov fp, sp              // make frame pointer pointing to current stack frame
16
17          ldr x0, =prompt         // load address of prompt string
18          bl printf               // call printf("\nEnter a number: ")
19          ldr x0, =frmt_int       // load address of format string
20          ldr x1, =number         // load address of int variable
21          bl scanf                // call scanf("%d",&n)
22          ldr x0, =msg_label      // load address of format string
23          ldr x1, =number         // load address of int variable
24          ldr x1, [x1]            // load int variable
25          bl printf               // call printf("You entered %d\n",n)
26          mov sp, fp              // make stack pointer pointing to the start of the current frame
27          ldp fp, lr,[sp], #16    // pop link register and the frame pointer from stack and shrink
28                                  // the stack by 16 bytes
29          mov x0, #0              // load return value
30          ret lr                  // return from callfunction
```

ARMv8 Rules for Calling a Function

- The first **8 parameters** are passed through the **registers x0 to x7**, the **first** through **x0,** the **second** through **x1,** and so on.
- If there are **more than 8 parameters**, they are **passed through the stack**, the last parameter is pushed first and so on, stack pointer is also expanded accordingly.
- The called function reads these parameters from the stack whenever necessary.
- The caller function needs to **pop the parameters** from the **stack** and shrink the stack pointer.

ARMv8 Passing 11 Parameters to printf() Function

```
1
2           .data
3   strfmt: .asciz "\nNumbers are: %d, %d, %d, %d, %d, %d, %d, %d, %d, %d\n"
4
5           .text
6           .globl passparameters
7   passparameters:
8           stp fp, lr, [sp, #-16]! // expand stack and push lr and fp onto stack
9           mov fp, sp              // make fp pointing the start of the current frame
10          ldr x0, =strfmt         // load address of format string
11                                  // to pass first parameter through x0
12          mov x1, #10             // Pass the second parameter through x1
13          mov x2, #20             // Pass the third parameter through x2
14          mov x3, #30             // Pass the fourht parameter through x3
15          mov x4, #40             // Passs the fifth parameter through x4
16          mov x5, #50             // Pass the sixth parameter through x5
17          mov x6, #60             // Pass the seventh parameter through x6
18          mov x7, #70             // Pass the eighth parameter through x7
19
20          mov x8, #80             // Use x8 as a scratch register
21          mov x9, #90             // Use x9 as a scratch register
22          mov x10, #100           // Use x10 as a scratch register
23          sub sp, sp, #32         // Grow stack to pass 9th, 10th, and 11th parameter
24          str x10, [fp, #-16]     // Pass the eleventh parameter through stack
25          str x9, [fp, #-24]      // Pass the tenth oarameter through stack
26          str x8, [fp, #-32]      // Pass the ninth parameter through stack
27          bl printf               // call printf with 11 parameters
28          mov sp, fp              // Adjust stack pointer upon return from printf
29          ldp fp, lr,[sp], #16    // pop saved lp and fp value from stack and shrink it
30          mov x0, #0              // load return value
31          ret lr                  // return from function passparameter
```

ARMv8 Function or Subroutine Writing Rules

- Assumes the **first eight parameters** have been passed through x0-x7 registers.
- **Additional parameters** are assumed on **stack** and can be accessed with **ldr x_n, [fp, #16], ldr x_n, [fp, #24]** but does not need to remove them from the stack.
- Registers x9-x15 **are preferred** for **local variables** or **scratch registers**, these are temporary registers and their contents are not preserved across function calls and returns. Registers x0-x7 can also be used for local variables or scratch registers but not recommended.

ARMv8 Function or Subroutine Writing Rules

- If the function needs to call some other functions and if it needs to preserve some values across these function calls, the **saved registers** x19-x27 should be used for this purpose.
- The **saved registers** current values must be pushed onto the stack before using them and must be popped back before returning from the function.
- If available registers are not enough for local variables, **local variables** should be implemented onto **stack**.
- If a local variables, such as **array** and **structure**, cannot fit into a register, they should be implemented onto **stack**.

ARMv8 Function or Subroutine Writing Rules

- If the function is going to **call some other functions**, the current lr and fp values must be saved on the stack and fp must point to the current stack frame.
- If the function is going to **use stack** either for **local variables** or for **passing parameters** to some other functions, the current lr and fp values must be saved on the stack, fp must point to the current stack frame, and the stack must be expanded to accommodate the local variables or the parameters.
- If the stack has **been expanded** either for local variables or for parameter passing, it **must be shrunk** appropriately.

ARMv8 Function or Subroutine Writing Rules

- If the lr and fp values have been saved on the stack for a valid reason, they must be retrieved and the stack must be shrunk appropriately.
- The **return value(s)** must be placed in x0 (x1-x7)
- Use ret lr statement at the end of the function body to **return** from the function

ARMv8 Function Writing Rules

```
#include <stdio.h>

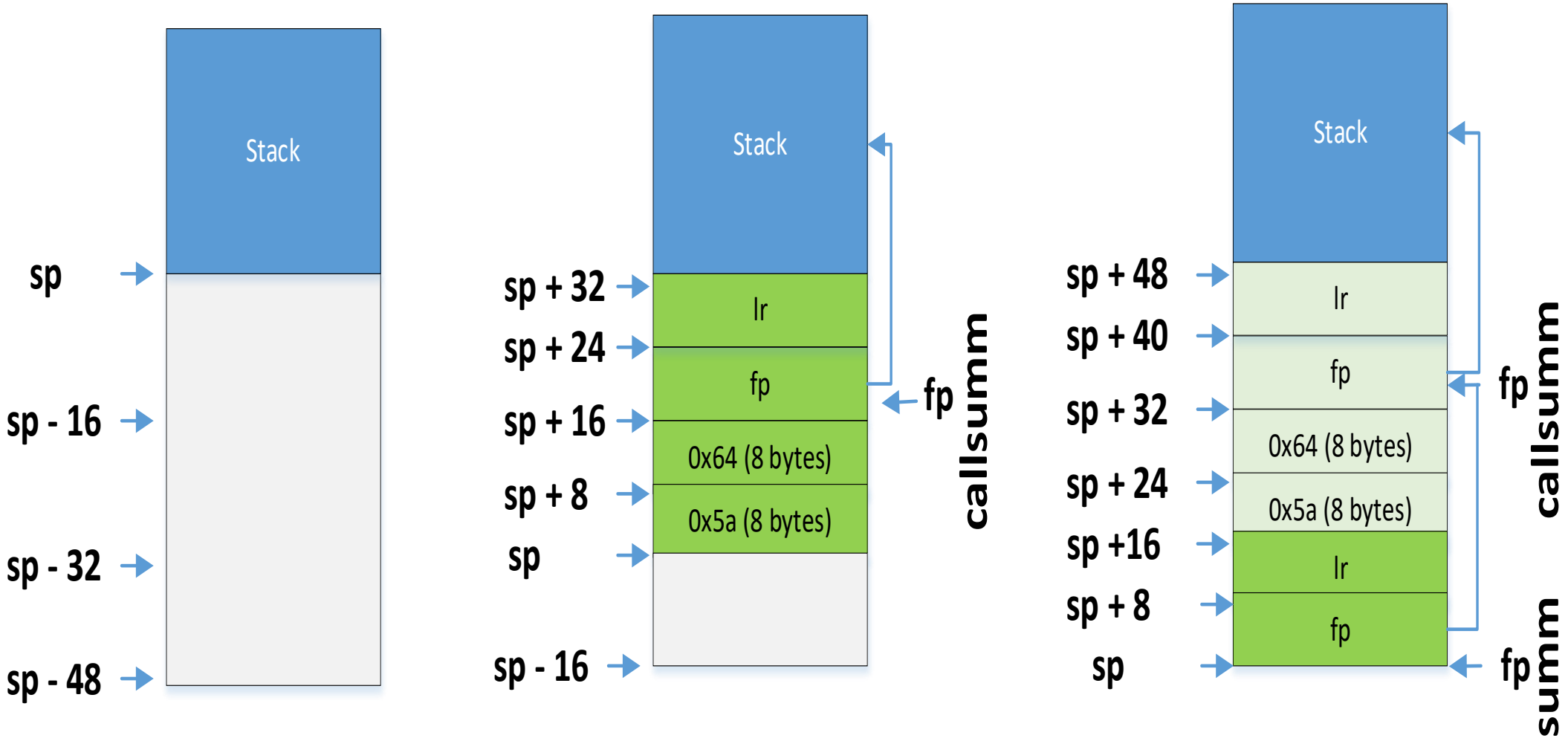
extern void summ(int n1, int n2, int n3, int n4, int n5,
                 int n6, int n7, int n8, int n9, int n10) {
                         int sum = n1+n2+n3+n4+n5+n6+n7+n8+n9+n10;
                         printf("Summation = %d\n",sum);
                 }

int callsumm() {
        summ(10, 20, 30, 40, 50, 60, 70, 80, 90, 100);
        return 0;
}
```

ARMv8 Function Writing Rules

```
27
28 callsumm:    stp fp, lr, [sp, #-16]!     //Expand stack 16 bytes and save lr and fp on stack
29              mov fp, sp                  //Make fp pointing to the current stack frame
30              sub sp, sp, #16             //Expand stack 16 bytes for passing 2 parameters
31              mov w0, #0xa                //Pass 10 through w0 register
32              mov w1, #0x14               //Pass 20 through w1 register
33              mov w2, #0x1e               //Pass 30 through w2 register
34              mov w3, #0x28               //Pass 40 through w3 register
35              mov w4, #0x32               //Pass 50 through w4 register
36              mov w5, #0x3c               //Pass 60 through w5 register
37              mov w6, #0x46               //Pass 70 through w6 register
38              mov w7, #0x50               //Pass 80 through w7 register
39              mov w8, #0x5a               //Load 90 into w8 register
40              mov w9, #0x64               //Load 100 into w9 register
41              str w9, [fp, #-8]           //Pass 100 in 8 bytes on stack at address sp + 8
42              str w8, [fp, #-16]          //Pass 90 in 8 bytes on stack at address sp + 0
43              bl summ                     //Call function summ
44              mov sp, fp                  //Shrink stack towards the frame pointer
45              ldp fp, lr, [sp], #16       //Load return address and frame pointer and shrink the stack
46              mov w0, wzr                 //Return zero to the caller
47              ret lr
48
```

ARMv8 Function Writing Rules



ARMv8 Function Writing Rules

```
1               .data
2  sumfmt:      .asciz "Summation of [10, 20, ...., 100] = %d\n"
3
4               .text
5               .global summ, callsumm
6
7  summ:        stp fp, lr, [sp, #-16]!     //Expand stack 16 bytes and save lr and fp on stack
8               mov fp, sp                  //Make fp pointing to the current stack frame
9               add w0, w0, w1              // w0 = w0 + w1
10              add w0, w0, w2              // w0 = w0 + w2
11              add w0, w0, w3              // w0 = w0 + w3
12              add w0, w0, w4              // w0 = w0 + w4
13              add w0, w0, w5              // w0 = w0 + w5
14              add w0, w0, w6              // w0 = w0 + w6
15              add w0, w0, w7              // w0 = w0 + w7
16              ldr w8, [fp, #24]           //Load 9th parameter from the stack into w8
17              add w0, w0, w8              // w0 = w0 + w8
18              ldr w9, [fp, #16]           //Load 10 parameter from the stack into w9
19              add w0, w0, w9              // w0 = w0 + w9
20              mov w1, w0                  // move summation result to w1 to pass to printf
21              ldr w0, =sumfmt             //Pass the address of strfmt to printf as the first parameter
22              bl printf                   //Call printf function
23              mov sp, fp                  //Shrink stack towards the frame pointer
24              ldp fp, lr, [sp], #16       //Load return address and frame pointer and shrink the stack
25              ret lr                      //Return from summ function
26
```

ARMv8 Function Writing: Using Pointer

```
char *str="NANAIMO";

int callpointer() {
   printf(str);
   reverseiterative(str, str+strlen(str)-1);
   printf(str);
   return 0;
}
```

```
void reverseiterative(char *left, char *right)  {
            char tmp;
            while(left<=right) {
                        tmp=*left;
                        *left=*right;
                        *right=tmp;
                        left++;
                        right--;
            }
}
```

ARMv8 Function Writing: Using Pointer

```
1
2               .data
3 nanaimo:      .asciz "\nNANAIMO\n"
4
5               .text
6               .global callpointer
7
8 callpointer:
9           stp fp, lr, [sp, #-16]! //Expand stack and save the return address and
10                                  //frame pointer on the stack
11          mov fp, sp              //Make frame pointer pointing the current frame
12          ldr x0, =nanaimo        //Call printf function to print nanaimo
13          bl printf
14          ldr x0, =nanaimo        //Call strlen function to get the length of nanaimo
15          bl strlen
16          sub x1, x0, #1          //Compute the index of the last character in nanaimo in x1
17          ldr x0, =nanaimo        //Load the pointer to the first character in nanaimo into x0
18          add x1, x0, x1          //Assign the pointer to the last character in nanaimo to x1
19          bl reverseiterative     //Call reverseiterative function
20          ldr x0, =nanaimo
21          bl printf               //Call printf function to print reversed nanaimo
22          mov sp, fp
23          ldp fp, lr, [sp], #16   //Retrieve saved return address and frame pointer from the
24                                  //stack and shrink the stack
25          mov x0, #0
26          ret lr                  //Return from callpointer function
27
```

ARMv8 Function Writing: Using Pointer

```
1
2           .text
3           .global reverseiterative
4
5   reverseiterative:
6
7   while:
8           cmp x0, x1              //compare pointer values
9           bge return              //if left >= right return
10          ldrb w3, [x0]           //load character pointed by left to w3
11          ldrb w4, [x1]           //load character pointed by right to w4
12          strb w3, [x1]           //store character in w3 to the byte pointed by right
13          strb w4, [x0]           //store character in w4 to the byte pointed by left
14          add x0, x0, #1          //increment left pointer
15          sub x1, x1, #1          //decrement right pointer
16          b while
17
18  return: ret lr
19
```

ARMv8 Function Writing: Global Array

```c

#include <stdio.h>

#define ARRAY_SIZE 10

const char* newline="\n";
const char* elmfmt ="%d ";

int source[] = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100};
int destination[ARRAY_SIZE];

void copy(int dst[], int src[], int size) {
    for(int i=0; i<size; i++) {
        dst[i] = src[i];
    }
}

void show(int arr[], int size) {
    for(int i=0; i<size; i++) {
        printf(elmfmt, arr[i]);
    }
}

int globalarray() {
    printf(newline);
    show(source, ARRAY_SIZE);
    printf(newline);
    copy(destination, source, ARRAY_SIZE);
    show(destination, ARRAY_SIZE);
    printf(newline);
    return 0;
}
```

ARMv8 Function Writing: Global Array

```
1
2                   .equ ARRAY_SIZE, 10
3                   .equ DATA_SIZE, 4
4
5                   .data
6   newline:        .asciz "\n"
7   elmfmt:         .asciz "%d "
8                   //Source array with initial values
9                   .align 2, 0
10  array_src:      .word 10, 20, 30, 40, 50, 60, 70, 80, 90, 100
11                  //Destination array initialized with zeros
12                  .align 2, 0
13  array_dest:     .skip ARRAY_SIZE*DATA_SIZE, 0
14
15
16                  .text
17                  .global showarray, copyarray, globalarray
18
19  showarray:
20
21  copyarray:
22
23  globalarray:
24
```

Continue…..

ARMv8 Function Writing: Global Array

Continued…..

```
62
63 globalarray:
64              stp fp, lr, [sp, #-16]! //Expand stack pointer and save lr and fp
65              mov fp, sp              //Make fp pointing current stack frame
66              ldr x0, =newline        //Print a new line
67              bl printf
68              ldr x0, =array_src      //Display source array
69              mov x1, ARRAY_SIZE      //by calling show function
70              bl showarray
71              ldr x0, =newline        //Print a new line
72              bl printf
73
74              ldr x0, =array_dest     //Copy source array
75              ldr x1, =array_src      //into destination array
76              mov w2, ARRAY_SIZE      //by calling copy function
77              bl copyarray
78
79              ldr x0, =array_dest     //Display destination array
80              mov x1, ARRAY_SIZE      //by calling show function
81              bl showarray
82              ldr x0, =newline        //Print a new line
83              bl printf
84
85              mov sp, fp              //Make sp pointing to current stack frame
86              ldp fp, lr, [sp], #16   //Load lr and fo and shrink stack
87              mov x0, xzr             //Return zero to the caller
88              ret lr
89
```

ARMv8 Function Writing: Global Array

Continued…..

```
18
19 showarray:
20              stp fp, lr, [sp, #-16]! //Expand the stack and save lr and fp on stack
21              mov fp, sp              //Make fp pointing to current stack frame
22              sub sp, sp, #16         //Expand stack to save array address, size, and index
23              str x0, [fp, #-8]       //Save the array address on stack
24              str w1, [fp, #-12]      //Save the array size on stack
25              mov w2, wzr             //Initialize array index to 0
26              str w2, [fp, #-16]      //Save the array index on stack
27 for_show:
28              ldr w2, [fp, #-16]      //Reload saved array index
29              ldr w1, [fp, #-12]      //Reload saved array size
30              cmp w2, w1              //Compare array index with array size
31              bge endfor_show
32              ldr x0, =elmfmt         //Load format string address onto x0 for printf
33              ldr x1, [fp, #-8]       //Load the saved array address into x1
34              ldr w1, [x1, x2, lsl#2] //Load the array element into x1 for printf
35              add w2, w2, #1          //Increment array index for the next iteration
36              str w2, [fp, #-16]      //Save incremented array index onto stack
37              bl printf               //Call printf passing format string through x0 and
38                                      //array element through x1
39              b for_show                  //Loop if the index is less than the size
40 endfor_show:
41              mov sp, fp              //Make stack pointer pointing current stack frame
42              ldp fp, lr, [sp], #16   //Load lr and fp and shrink the stack
43              ret lr
44
```

Continued…..

ARMv8 Function Writing: Global Array

```
46
47 copyarray:
48                                  //x0 source array address
49                                  //x1 destination array address
50                                  //x2 array size
51          mov x3, #0              //Initialize array index
52 for_copy:
53          cmp x3,x2               //Campre array index against its size
54          bge endfor_copy
55          ldr w4,[x1, x3, lsl#2]  //Load array element from source
56          str w4,[x0, x3, lsl#2]  //Save array element onto destination
57          add x3, x3, #1          //Incement array index
58          b for_copy              //Loop if the index is less than size
59 endfor_copy:
60          ret lr
61
```

ARMv8 Function Writing: Local Array

```c

#include <stdio.h>

#define ARRAY_SIZE 10

const char* newline="\n";
const char* elmfmt ="%d ";

int source[] = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100};

void copy(int dst[], int src[], int size) {
    for(int i=0; i<size; i++) {
        dst[i] = src[i];
    }
}

void show(int arr[], int size) {
    for(int i=0; i<size; i++) {
        printf(elmfmt, arr[i]);
    }
}

int localarray() {
    int destination[ARRAY_SIZE];
    printf(newline);
    show(source, ARRAY_SIZE);
    printf(newline);
    copy(destination, source, ARRAY_SIZE);
    show(destination, ARRAY_SIZE);
    printf(newline);
    return 0;
}
```

ARMv8 Function Writing: Local Array

```
1
2           .set ARRAY_SIZE, 10
3           .set ARRAY_ELEMENT_SIZE, 4
4           .set LOCAL_ARRAY_BYTES, ARRAY_SIZE*ARRAY_ELEMENT_SIZE
5
6           .data
7 newline:  .asciz "\n"
8           .align 3, 0
9 larrfmt:  .asciz "Local Array Sum Result: %d\n"
10          //Source array with initial values
11          .align 2, 0
12 array_src: .word 10, 20, 30, 40, 50, 60, 70, 80, 90, 100
13
14          .text
15          .global localarray
16
17
18 localarray:
```

ARMv8 Function Writing: Local Array

Continued…..

```
17
18 localarray:
19              stp fp, lr, [sp, #-16]!
20              mov fp, sp
21              //LOCAL_ARRAY_BYTES represents the total byte requirement
22              //for the local array, it might not be multiple of 16.
23              //Local array is implemented on the stack and its expansion
24              //must be scaled 16.
25              mov w0, LOCAL_ARRAY_BYTES    //Load LOCAL_ARRAY_BYTES onto w0
26              mov w1, #0xf                 //Load the last 4-bit mask 0x000f into w1
27              and w2,w0,w1                 //Mask the last 4 bits
28              cmp w2, wzr                  //Compare with zeros
29              beq _16_multiple                 //Multiple of 16 if the last 4 bits are zeros
30 make_16_multiple:
31              //Compute next multiple of 16 vlaue from LOCAL_ARRAY_BYTES value
32              mov w1, #0xfff0              //Load 12 bit mask into w1
33              and w0, w0, w1               //Make w0 floor of multiple of 16
34              add w0, w0, #16              //Make w0 ceiling of multiple of 16
35
36 _16_multiple:
37              sub sp, sp, x0          //Expand stack pointer downward
38                                      //to accomodate LOCAL_ARRAY_SIZE integers
39
```

ARMv8 Function Writing: Local Array

Continued…..

```
36 _16_multiple:
37             sub sp, sp, x0          //Expand stack pointer downward
38                                     //to accomodate LOCAL_ARRAY_SIZE integers
39
40             ldr x0, =newline        //Print a new line
41             bl printf
42             ldr x0, =array_src      //Display source array
43             mov x1, ARRAY_SIZE      //by calling show function
44             bl showarray
45             ldr x0, =newline        //Print a new line
46             bl printf
47
48             mov x0, sp              //Copy source array
49             ldr x1, =array_src      //into destination array
50             mov w2, ARRAY_SIZE      //by calling copy function
51             bl copyarray
52
53             mov x0, sp              //Display destination array
54             mov x1, ARRAY_SIZE      //by calling show function
55             bl showarray
56             ldr x0, =newline        //Print a new line
57             bl printf
58
59             mov sp, fp              //Make sp pointing to current stack frame
60             ldp fp, lr, [sp], #16   //Load lr and fp and shrink stack
61             mov x0, xzr             //Return zero to the caller
62             ret lr
63
```

ARMv8 Function Writing: Recursion

```
#include <stdio.h>

int factorial(int n) {
        if(n=<1) return 1;
        return n*factorial(n-1);
}

int recursion_factorial() {
    printf("\nFactorial of %d is %d", 5, factorial(5));
    return 0;
}
```

Function Writing: Recursion

```
1         .data
2 strfmt: .asciz "\nFactorial of %d is %d"
3
4         .text
5         .global factorial, recursion_factorial
6
7 factorial:
8         cmp x0, #1              // Compare the function parameter with 1
9         ble base                // Function parameter is less than or equal to 1,
10                                // Jump to base case
11 recur:                         // Recursion part of factorial function
12        stp fp, lr, [sp, #-16]! // Expand stack and save lr and fp on stack
13        mov fp, sp              // Make fp pointing to current stack frame
14        sub sp, sp, #16         // Expand stack pointer to push current n,
15        str x0, [fp, #-16]      // Push the current n in x0 onto the stack at [fp, #-16] memory
16                                // address
17        sub x0, x0, #1          // Subtract 1 from the current n value to pass
18                                // (n-1) to the next recursive call of the function
19        bl factorial            // Call factorial function recursively
20                                // Recursive call returned the result in x0,
21        ldr x1, [fp, #-16]      // Pop the pushed n value from the stack into x1
22        mul x0, x0, x1          // Multiply the popped n with the returned result of
23                                // factorial(n-1) to compute factorial(n)
24        mov sp, fp              // Make sp pointing to current stack frame
25        ldp fp, lr, [sp], #16   // Pop the pushed lr and fp from the stack and
26                                // shrink the stack
27        ret lr                  // Return from the current call of factorial frunction with
28                                // the new intermediate result
29 base:   mov x0, #1             // Base case of factorial function, return 1 from the base case
30        ret lr                  // Return from factorial function
```

Function Writing: Recursion

```
32
33          .equ number, 5
34
35  recursion_factorial:
36          stp fp, lr, [sp, #-16]!
37          mov fp, sp
38          mov x0, number
39          bl factorial
40          mov x2, x0
41          mov x1, number
42          ldr x0, =strfmt
43          bl printf
44          mov sp, fp
45          ldp fp, lr, [sp], #16
46          ret lr
47
```

Function Writing: Recursion

```c

#include <stdio.h>

int fib_index = 10;

int fibonacci(int fib_index) {
    if (fib_index == 0 || fib_index == 1) {
        return fib_index;
    }
    return fibonacci(fib_index-2) + fibonacci(fib_index-1);
}

int recursion_fibonacci() {
    printf("\nFibonacci %d is %d", fib_index, fibonacci(fib_index));
    return 0;
}
```

Function Writing: Recursion

```
1           .data
2 fib_index:  .word 10
3 frmt_str:   .asciz "\nFibonacci %d is %d\n"
4
5           .text
6           .global fibonacci, recursion_fibonacci
7
8 fibonacci:
9           cmp x0, xzr             //Compare n with zero
10          beq base                //If zero, branch to base case to return zero
11          cmp x0, #1              //Compare n with one
12          beq base                //If one, branch to base case to return one
13
14 recur:    stp fp, lr, [sp, #-16]! //Grow stack pointer and save lr and fp on stack
15          mov fp, sp              //Make fp pointing to current stack frame
16          sub sp, sp, #16         //Grow stack to save x19, and x20 registers content
17          str x19, [fp, #-8]      //Save x19 register on the stack
18          str x20, [fp, #-16]     //Save x20 register on the stack
19          mov x19, x0             //Save current fib_index in x19
20          sub x0, x0, #1          //Call fibonacci function recursivle with (n -1)
21          bl fibonacci
22          mov x20, x0;            //Save the fib(n -1) into x20
23          mov x0, x19             //Call fibonacci function recursively with (n -2)
24          sub x0, x0, #2
25          bl fibonacci
26          add x0, x0, x20         //Add fib(n -1) with fib(n -2) to compute fib(n)
27          ldr x20, [fp, #-16]     //Retrieve saved x20 value from the stack
28          ldr x19, [fp, #-8]      //Retrieve saved x19 value from stack
29          mov sp, fp              //Shrink stack pointer appropriately
30          ldp fp, lr, [sp], #16   //Retrieve the saved lr and fp from stack and shrink it
31 base:     ret lr                  //Return from fibonacci function
```

Function Writing: Recursion

```
32
33
34 recursion_fibonacci:
35
36          stp fp, lr, [sp,#-16]!  //Grow stack pointer and save lr and fp on stack
37          mov fp, sp              //Make fp pointing tp the current stack frame
38          ldr x0, =fib_index      //Load fib_index address into x0
39          ldr w0, [x0]            //Load fib_index value into w0
40          bl fibonacci            //Call fibonacci function passing fib_index
41          mov x2, x0              //Move function result from x0 to x2
42          ldr x1, =fib_index      //Load fib_index address into x0
43          ldr w1, [x1]            //Load fib_index value into w1
44          ldr x0, =frmt_str       //Load format string addrees into x0
45          bl printf               //Call printf passing format string address,
46                                  //fib_index, and fibonacci result
47          mov sp, fp
48          ldp fp, lr, [sp], #16   //Retrieve the save lr and fp from stack and shrink it
49          ret lr                  //Return from the function
50
```

ARMv8 Function Writing: Recursion Using Pointer

```
char *str="NANAIMO";

int callpointer() {
   printf(str);
   reverserecursive(str, str+strlen(str)-1);
   printf(str);
   return 0;
}
```

```
void reverserecursive(char *left, char *right)  {
    char tmp;
    if(left<=right) {
            tmp=*left;
            *left=*right;
            *right=tmp;
            reverserecursive(left+1,right-1);
     }
}
```

Using C Global Structure

```c

#include <stdio.h>
#include <string.h>

struct student_st {
    char first_name[30];
    char last_name[30];
    unsigned char class;
    int grade;
    };

struct student_st student;

int globalstructure() {
    strcpy(student.first_name, "Humayun");
    strcpy(student.last_name, "Kabir"));
    student.class = 5;
    student.grade = 100;
    printf("\nStudent: [%s, %s, %d, %d]\n",
            student.first_name, student.last_name, student.class, student.grade);
    return 0;
    }

```

ARMv8 Using C Global Structure

```
1
2               .equ st_fn_offset, 0        //The start of the structure is scaled 8
3               .equ st_ln_offset, 30       //does not need to be scaled, strcpy copies byte by byte
4               .equ st_class_offset, 60    //does not need to be scaled for strb wn, []
5               .equ st_grade_offset, 64    //needs to be scaled 4 for str wn, []
6               .equ struct_size, 68        //64+4 = 68
7
8               .data
9   fname:      .asciz "Humayun"
10              .align 3, 0
11  lname:      .asciz "Kabir"
12              .align 3, 0
13  strfmt:     .asciz "\nStudent: [%s %s, %d, %d]\n"
14
15              .align 3, 0
16  student:    .skip struct_size, 0
17
18              .text
19              .global globalstructure
20
21  globalstructure:
```

Continue……

..... Continued

ARMv8 Using C Global Structure

```
20
21 globalstructure:
22              stp fp,lr, [sp, #-16]!      //Adjust the stack and save lr and fp
23              mov fp, sp                  //Make fp pointing to current stack frame
24              ldr x0, =student            //Load student structure address into x0
25              add x0, x0, st_fn_offset    //Add first name field offset with the address
26              ldr x1, =fname              //Load the addres of first name string into x1
27              bl strcpy                   //Copy the string into the fist name field of the structure
28              ldr x0, =student            //Load student structure address into x0
29              add x0, x0, st_ln_offset    //Add last name field offset with the address
30              ldr x1, =lname              //Load the address of the last name string into x1
31              bl strcpy                   //Copy the string into the last name field of the structure
32              ldr x0, =student                //Load student structure address into x0
33              mov w8,#5                       //Initialize w8 with a class value 5
34              strb w8, [x0, st_class_offset]  //Save class value 5 at class field of the structure
35              mov w8, #100                    //Initialize w8 with grade value 100
36              str w8, [x0, st_grade_offset]   //Save grade value 100 at grade field of the structure
37              ldr w4, [x0, st_grade_offset]   //Load the student grade into w4
38              ldrb w3, [x0, st_class_offset]  //Load the student class int0 w3
39              mov x2, x0                      //Set the student structue address into x2 from x0
40              add x2, x2, st_ln_offset        //Add the last name field offset to the address
41              mov x1, x0                      //Set the student structure address into x1 from x0
42              add x1, x1, st_fn_offset        //Add fist name field offset to the address
43              ldr x0, =strfmt                 //Load print format string address into x0
44              bl printf                       //Call printf to print all the fields of the student
45              mov sp, fp
46              ldp fp, lr, [sp], #16           //Load lr and fp from stack and shrink it
47              mov x0, xzr
48              ret lr
49
```

Using C Local Structure

```c

#include <stdio.h>
#include <string.h>

struct student_st {
    char first_name[30];
    char last_name[30];
    unsigned char class;
    int grade;
    };

int localstructure() {
    struct student_st student;
    strcpy(student.first_name, "Humayun");
    strcpy(student.last_name, "Kabir"));
    student.class = 5;
    student.grade = 100;
    printf("\nStudent: [%s, %s, %d, %d]\n",
            student.first_name, student.last_name, student.class, student.grade);
    return 0;
    }

```

ARMv8 Using C Local Structure

```
1
2           .equ st_fn_offset, 0        //The start of the structure is scaled 8
3           .equ st_ln_offset, 30       //does not need to be scaled, strcpy copies byte by byte
4           .equ st_class_offset, 60    //does not need to be scaled for strb wn, []
5           .equ st_grade_offset, 64    //needs to be scaled 4 for str wn, []
6           .equ struct_size, 80        //64+4 = 68, next multiple of 16 is 80
7
8
9           .data
10 fname:   .asciz "Humayun"
11 lname:   .asciz "Kabir"
12 strfmt:  .asciz "\nStudent: [%s %s, %d, %d]\n"
13
14          .text
15          .global localstructure
16
17 localstructure:
```

Continue……

..... Continued

ARMv8 Using C Local Structure

```
16
17 localstructure:
18         stp fp, lr, [sp, #-16]!          //Expand stack and save lr and fp
19         mov fp, sp                       //Make fp pointing to current stack frame
20         sub sp, sp, struct_size          //Exapnd the stack pointer for local student structure
21         mov x0, sp                       //Set the local student structure address into x0
22         add x0, x0, st_fn_offset         //Add first name field offset with the address
23         ldr x1, =fname                   //Load the address of the first name string into x1
24         bl strcpy                        //Copy the string into the first name field of the structure
25         mov x0, sp                       //Set the local student structure address into x0
26         add x0, x0, st_ln_offset         //Add last name field offset to the address
27         ldr x1, =lname                   //Load the address of the last name string into x1
28         bl strcpy                        //Copy the string into the last name field of the structure
29         mov w8,#5                        //Initialize w8 with class value 5
30         strb w8, [sp, st_class_offset]   //Save class value 5 at the class field of the stucture
31         mov w8, #100                     //Initialize w8 with grade value 100
32         str w8, [sp, st_grade_offset]    //Save grade value 100 at the grade field of the structure
33
34         ldr x0, =strfmt                  //Load print format string into x0
35         mov x1, sp                       //Set local student structure address into x1
36         add x1, x1, st_fn_offset         //Add first name field offset to the address
37         mov x2, sp                       //Set local student structure address into x2
38         add x2, x2, st_ln_offset         //Add last name field offset to the address
39         ldrb w3, [sp, st_class_offset]   //Load the class value of student into w3
40         ldr w4, [sp, st_grade_offset]    //Load the grade value of student into w4
41         bl printf                        //Call printf
42         mov sp, fp
43         ldp fp, lr, [sp], #16            //Load lr and fp from stack and shrink it
44         mov w0, wzr
45         ret lr
46
```

ARMv8 Scaled Address

```
1           .data
2  num1:    .word 10
3           .align 3, 0
4  num2:    .word -20
5           .align 3, 0
6  num3:    .word 30
7           .align 3, 0
8  num4:    .word -40
9           .align 3, 0
10 num5:    .word 50
11          .align 3, 0
12 strfmt:  .asciz "Current number: %d\n"
13
14
15          .text
16          .global scaled
17
18 scaled:
19          stp fp, lr, [sp, #-16]
20          mov fp, sp
21
22          ldr x1, =num1
23          ldr x1, [x1]
24          ldr x0, =strfmt
25          bl printf
26
```

```
26
27          ldr x1, =num2
28          ldr x1, [x1]
29          ldr x0, =strfmt
30          bl printf
31
32          ldr x1, =num3
33          ldr x1, [x1]
34          ldr x0, =strfmt
35          bl printf
36
37          ldr x1, =num4
38          ldr x1, [x1]
39          ldr x0, =strfmt
40          bl printf
41
42          ldr x1, =num5
43          ldr x1, [x1]
44          ldr x0, =strfmt
45          bl printf
46
47          mov sp, fp
48          ldp fp, lr, [sp], #16
49          mov x0, xzr
50          ret lr
51
```

ARMv8 Unscaled Address

```
1               .data
2   num1:       .word -10
3   num2:       .word 20
4   num3:       .word -30
5   num4:       .word 40
6   num5:       .word -50
7   strfmt:     .asciz "Current number: %d\n"
8
9
10              .text
11              .global unscaled
12
13  unscaled:
14              stp fp, lr, [sp, #-16]!
15              mov fp, sp
16
17              ldr x1, =num1
18              ldursw x1, [x1]
19              ldr w0, =strfmt
20              bl printf
21
22              ldr x1, =num2
23              ldursw x1, [x1]
24              ldr x0, =strfmt
25              bl printf
26
```

```
26
27              ldr x1, =num3
28              ldursw x1, [x1]
29              ldr x0, =strfmt
30              bl printf
31
32              ldr x1, =num4
33              ldursw x1, [x1]
34              ldr x0, =strfmt
35              bl printf
36
37              ldr x1, =num5
38              ldursw x1, [x1]
39              ldr x0, =strfmt
40              bl printf
41
42              mov sp, fp
43              ldp fp, lr, [sp], #16
44              mov x0, xzr
45              ret lr
46
```