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 in the code.

```
                .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 x0, =number1        //main function starts
                ldr x0, [x0]
                ldr x1, =number2
                ldr x1, [x1]
                ldr x3, =number3
                ldr x3, [x3]            //Codes for main function
                add x4, x0, x1
                add x4, x4, x3
                mov x0, x4
                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.

ARMv8 Assembly Programming: Basics

```
1           .data                   //Data segment of the program starts here
2 greet:    .asciz "Hello World!\n" //Defines a null terminated string named 'greet' in data segment
3                                   //A blank line between data and code segments
4           .text                   //Code segment of the program starts here
5           .global main            //Makes the function 'main' global
6 main:     str lr, [sp, #-16]!     //Code for function 'main' in code segment starts here, stores the return address onto stack
7           ldr x0, =greet          //Loads the address of 'greet' string into x0 register
8           bl printf               //Calls 'printf' function passing the string address as function parameter through register x0
9           mov x0, xzr             //Initializes x0 register to zero in order to return zero from 'main' function
10          ldr lr, [sp], #16       //Loads return address from the stack into 'lr'
11          ret lr                  //Returns from function 'main'
```

[label:] [directive or instruction] // comment

Selection C Code

```
static int a = 10;
static int b = 4;
static int x;

int main()  {
        if ( a < b )
                x = a;
        else
                x = b;
        return 0;
}
```

Selection ARMv8 Assembly Code

```
1           .data
2   a:      .word 10          // static int a=10;
3           .skip 4, 0
4   b:      .word 4           // static int b=4;
5           .skip 4, 0
6   x:      .word 0           // static int x;
7
8           .text
9           .globl main
10  main:   ldr x0, =a        // load address of 'a'
11          ldr x1, =b        // load address of 'b'
12          ldr x0,[x0]       // load 'a'
13          ldr x1,[x1]       // load 'b'
14          cmp x0, x1        // compare them
15          blt done          // if a < b then goto done; x0 has the smaller of two
16  else:   mov x0, x1        // else bring smaller of two into x0
17  done:   ldr x1, =x        // load pointer to 'x'
18          str x0, [x1]      // store x0 in 'x'
19          mov x0, #0
20          ret lr
```

ARMv8 Rules for Calling a Function

- Caller function needs to **push** the **link register** current value onto the **stack** and adjust the stack pointer accordingly.
- **Stack address grows downward** and it must be **minimum 16 scaled**, i.e., stack pointer should be **subtracted** at **least 16** to grow.
- First **8 parameters** are passed through the **registers x0 to x7**, the **first** through **x0,** the **second** through **x1,** and so on.
- Before return caller function **pops the link register value** from the **stack** into ***lr*** register and adjust the stack pointer accordingly.
- **Stack address shrink upward**, i.e., should be added at least **16**.

ARMv8 Calling C Library Functions

```
1           .data
2   str0:   .asciz "\nEnter a number: "
3           .align 3
4   str1:   .asciz "%d"
5           .align 3
6   str2:   .asciz "You entered %d\n"
7   n:      .long 0
8
9           .text
10          .globl main
11  main:
12          str lr, [sp, #-16]! // push link register onto stack
13          ldr x0, =str0       // load address of prompt string
14          bl printf           // call printf("\nEnter a number: ")
15          ldr x0, =str1       // load address of format string
16          ldr x1, =n          // load address of int variable
17          bl scanf            // call scanf("%d",&n)
18          ldr x0, =str2       // load address of format string
19          ldr x1, =n          // load address of int variable
20          ldr x1, [x1]        // load int variable
21          bl printf           // call printf("You entered %d\n",n)
22          mov x0, #0          // load return value
23          ldr lr,[sp], #16    // pop link register from stack
24          ret lr              // return from main
25
26
```

ARMv8 Rules for Calling a Function

- If there are **more than 8 parameters**, they are **pushed onto the stack**, the last parameter is pushed first an so on, stack pointer is also adjusted accordingly. Called function reads these parameters from the stack whenever necessary.
- If the parameters were pushed onto the stack to call a function, upon return from the called function the caller function needs to **pop the parameters** from the **stack** and adjust 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
6           .text
7           .globl main
8   main:
9           str lr, [sp, #-16]!         // push link register onto stack
10          ldr x0, =strfmt             // load address of format string
11                                      // to pass 1st parameter through x0
12          mov x1, #10                 // Pass the 2nd parameter through x1
13          mov x2, #20                 // Pass the 3rd parameter through x2
14          mov x3, #30                 // Pass the 4th parameter through x3
15          mov x4, #40                 // Pass the 5th parameter through x4
16          mov x5, #50                 // Pass the 6th parameter through x5
17          mov x6, #60                 // Pass the 7th parameter through x6
18          mov x7, #70                 // Pass the 8th parameter through x7
19          mov x8, #80                 // Use register x8 as the scratch register
20          mov x9, #90                 // Use register x9 as the scartch register
21          mov x10, #100               // Use register x10 as the scratch register
22          sub sp, sp, #32             // Grow stack to pass 9th, 10th, and 11th parameters
23          str x10, [sp, #16]          // Pass the 11th parameter first through the stack
24          str x9, [sp, #8]            // Pass the 10th parameter second through the stack
25          str x8, [sp]                // Pass the 9th parameter last through the stack
26          bl printf                   // call printf with 11 parameters
27          add sp, sp, #32             // Adjust stack pointer upon return from printf
28          mov x0, #0                  // load return value
29          ldr lr,[sp], #16            // pop link register value from stack
30          ret lr                      // return from main
```

Iteration C Code

```c
int main() {
        int sum = 0;
        int i;
        for(i=0; i<10; i++) {sum += i;}
        printf("Summation of [0,1,2,..,9] = %d\n", sum);
        return 0;
}
```

```c
int main() {
        int sum = 0;
        int i= 0;
        while(i<10) { sum += i;  i++; }
        printf("Summation of [0,1,2,..,9] = %d\n", sum);
        return 0;
}
```

Iteration ARMv8 Assembly Code

```
1
2               .data
3   strfmt:     .asciz "Summation of [0,1,2,..,9] = %d\n"
4
5               .text
6               .globl main
7   main:       sub sp, sp, #16 //Adjust stack pointer downward
8               str lr, [sp]    //Store return address onto stack
9               mov x1, xzr     // Initialize sum = 0
10              mov x2, #0      // use x2 for i; i=0
11  loop:       cmp x2, #10     // perform comparison
12              bge loop_exit   // end loop if i >= 10
13              add x1, x1, x2  // sum += i, sum +=i
14              add x2, x2, #1  // i++
15              b loop          // repeat loop test
16
17  loop_exit:  ldr x0, =strfmt //Load the address of stirng format
18                              //to pass as the first parameter to printf() function
19                              //sum value is already in x1, pass it as the second
20                              //parameter to printf() function
21              bl printf       //Call printf function
22
23              mov x0, xzr     //Initialize x0 to 0 to return 0 from main() function
24              ldr lr, [sp]    //Load return address from the stack
25              add sp, sp, #16 //Adjust stack pointer upward
26              ret lr          //Return from the main function
27
```

ARMv8 Function or Subroutine Writing Rules

- When writing a subroutine or function:
  - The **first eight parameters** are assumed in x0-x7
  - **Additional parameters** are assumed in **stack** and can be accessed with **ldr x_n, [sp, #offset]** but does not need to remove them from the stack.
  - Free to change the content of registers x0-x7
  - Registers x0-x7 can also be used for local variables.
  - Registers x9-x15 **are preferred** for **local variables**, these are temporary registers, their contents are not preserved across function call and return.

ARMv8 Function or Subroutine Writing Rules

- When writing a subroutine or function:
  - If the **saved registers** x19-x27 needed to be used inside the function for local variables, there current values must be pushed onto the stack and the pushed values must be popped before return.
  - If available registers are not enough for local variables, **local variables** are implemented onto **stack**.
  - If a local variable cannot fit into a register, it must be implemented onto stack. For example, local **array** and local **struct**.

ARMv8 Function or Subroutine Writing Rules

- When writing a subroutine or function:
  - If the function is going to **call another function**, rules for calling a function must be followed.
  - The **return value** must be placed in x0 (and possibly x1-x7)
  - The return address has already been copied into register lr when the function has been called by the caller.
  - Use ret lr statement at the end of the function body to **return** from the function

ARMv8 Function Writing Rules

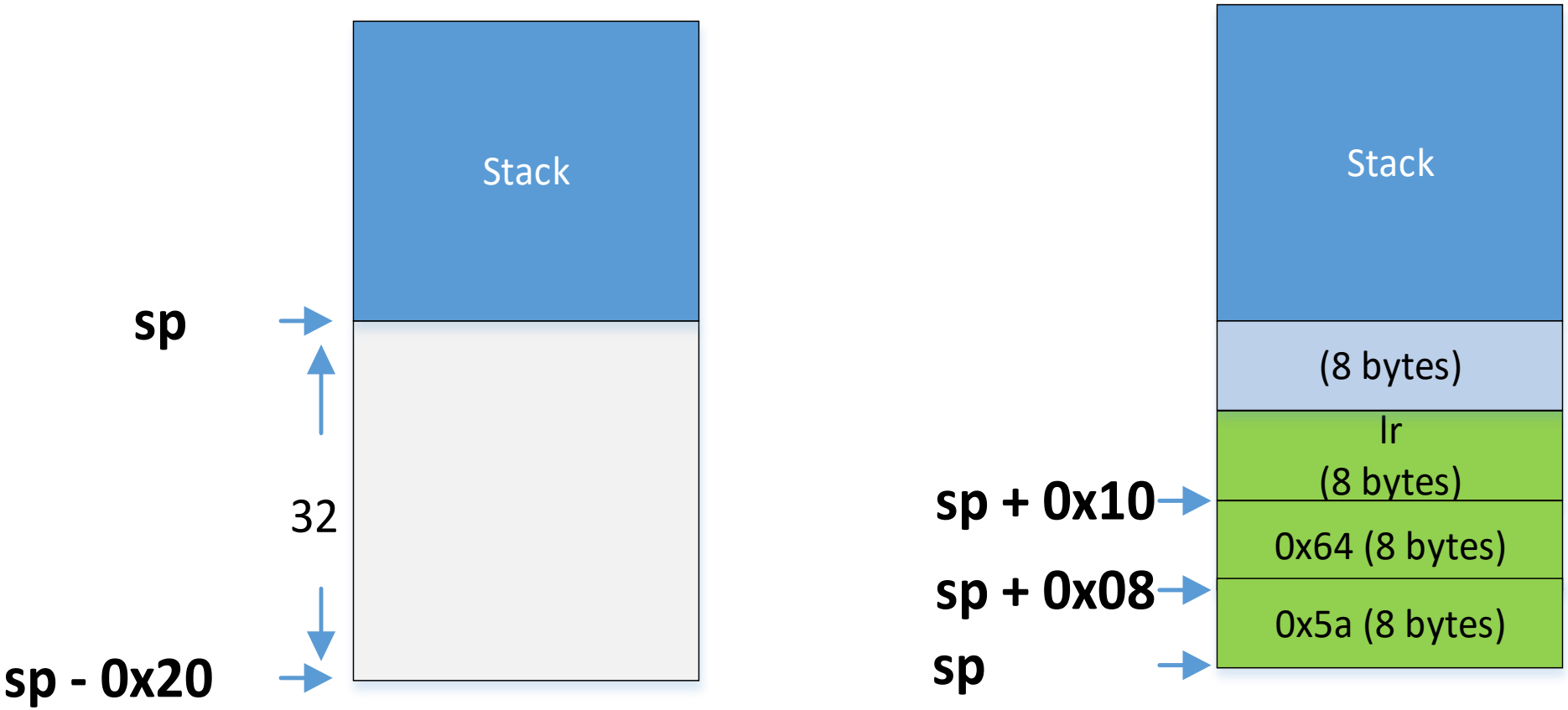
```
1
2  #include <stdio.h>
3
4  extern int summ(int n1, int n2, int n3, int n4, int n5,
5          int n6, int n7, int n8, int n9, int n10);
6
7  int main(void) {
8      /* prints result of sum() function */
9      printf("Summation = %d\n",
10             summ(10, 20, 30, 40, 50, 60, 70, 80, 90, 100));
11     return 0;
12 }
13
```

ARMv8 Function Writing Rules

```
1           .text
2           .global summ
3
4   summ:   add w0, w0, w1              // w0 = w0 + w1
5           add w0, w0, w2              // w0 = w0 + w2
6           add w0, w0, w3              // w0 = w0 + w3
7           add w0, w0, w4              // w0 = w0 + w4
8           add w0, w0, w5              // w0 = w0 + w5
9           add w0, w0, w6              // w0 = w0 + w6
10          add w0, w0, w7              // w0 = w0 + w7
11          ldr w8, [sp]                //Load 9th parameter from the stack into w8
12          add w0, w0, w8              // w0 = w0 + w8
13          ldr w9, [sp, #0x8]          //Load 10 parameter from the stack into w9
14          add w0, w0, w9              // w0 = w0 + w9
15          ret lr                      // Result is already in w0, return
16
17
```

```
1               .data
2   sumfmt:     .asciz "Summation of [10, 20, ...., 100] = %d\n"
3
4
5               .text
6               .globl main
7   main:       sub sp, sp, #0x20           //Adjusted the stack pointer 32 bytes downward
8                                           //to accommodate the return address and 2 parameters
9                                           //of summ() function
10              str lr, [sp, #0x10]         //Store the return addres from lr (8 bytes) on stack at
11                                          //address sp + 16
12              mov w0, #0xa                //Pass 10 through w0 register
13              mov w1, #0x14               //Pass 20 through w1 register
14              mov w2, #0x1e               //Pass 30 through w2 register
15              mov w3, #0x28               //Pass 40 through w3 register
16              mov w4, #0x32               //Pass 50 through w4 register
17              mov w5, #0x3c               //Pass 60 through w5 register
18              mov w6, #0x46               //Pass 70 through w6 register
19              mov w7, #0x50               //Pass 80 through w7 register
20              mov w8, #0x5a               //Load 90 into w8 register
21              mov w9, #0x64               //Load 100 into w9 register
22              str w9, [sp, #0x8]          //Pass 100 in 8 bytes on stack at address sp + 8
23              str w8, [sp]                //Pass 90 in 8 bytes on stack at address sp + 0
24              bl summ                     //Call summ() function
25              mov w1, w0                  //Retrieve result from summ function to pass it to printf()
26                                          //function as the second parameter
27              ldr w0, =sumfmt             //Pass the address of strfmt to printf() function as the
28                                          //first parameter
29              bl printf                   //Call printf function
30
31              ldr lr, [sp, #0x10]         //Load return address
32              add sp, sp, #0x20           //Readjust the stack pointer 32 bytes upward
33              mov x0, xzr                 //Return zero to the caller
34              ret lr
35
```

ARMv8 Function Writing Rules



ARMv8 Function Writing: Local Array

```
int larray() {
        int x[20];
        /* try to keep i in a register */
        register int i;
        for(i=0; i<20; i++) { x[i] = i; }
        register int sum = 0;
        for(i=19; i>=0; i--) { sum += x[i] ; }
        return sum;
}
```

ARMv8 Function Writing: Local Array

```
1
2               .text
3               .global larray
4
5  larray:      sub sp, sp, #160          //Adjust the stack pointer downward
6                                         //to accomodate 20 integers or int[20],
7                                         //8 bytes for each integer although an
8                                         //integer size is 4 bytes, 8 bytes are
9                                         //allocated to make the address sacled
10              mov x0, #0                //for(i=0; i<20; i++)
11 loop1:       cmp x0, #20
12              bge loop1_exit
13              str x0,[sp, x0, lsl#3]    //arr[i] = i
14              add x0, x0, #1
15              b loop1
16 loop1_exit:
17              sub x0, x0, #1            //i-- to make i=19
18              mov x1, #0;               //Use x1 for total = 0
19 loop2:       cmp x0, #0                //for (i=19; i>=0; i--)
20              blt loop2_exit
21              ldr x2, [sp, x0, lsl#3]
22              add x1, x1, x2            //total += arr[i]
23              sub x0, x0, #1
24              b loop2
25
26 loop2_exit:
27              mov x0, x1                //Return result through x0
28              add sp, sp, #160          //Adjust the stack pointer upward
29                                        //before returning from the function
30              ret lr
```

ARMv8 Function Writing: Local Array

```
1
2               .data
3
4   larrfmt:    .asciz "Local Array [0, 1, 2, ...., 19] Total: %d\n"
5
6
7               .text
8               .globl main
9   main:       str lr, [sp, #-16]!
10
11              bl larray
12              mov x1, x0                  //Retrieve return value from local function into x1
13                                          //in order to pass it to printf function
14              ldr x0, =larrfmt            //Load format string addredd into x0 to pass it
15                                          //to printf function
16              bl printf                   //Call printf function
17
18              ldr lr, [sp], #16           //Load return address
19              mov x0, xzr                 //Return zero to the caller
20              ret lr
21
```

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

ARMv8 Function Writing: Global Array

```
1
2                   .equ ARRAY_SIZE, 10
3                   .equ DATA_SIZE, 8
4
5                   .data
6   newline:        .asciz "\n"
7
8
9   //Source array with initial values
10                  .align 3, 0
11  source:         .word 10
12                  .skip 4, 0
13                  .word 20
14                  .skip 4, 0
15                  .word 30
16                  .skip 4, 0
17                  .word 40
18                  .skip 4, 0
19                  .word 50
20                  .skip 4, 0
21                  .word 60
22                  .skip 4, 0
23                  .word 70
24                  .skip 4, 0
25                  .word 80
26                  .skip 4, 0
27                  .word 90
28                  .skip 4, 0
29                  .word 100
30                  .skip 4, 0
31
32  //Destination array initialized with zeros
33                  .align 3, 0
34  destination:    .skip ARRAY_SIZE*DATA_SIZE, 0
35
36
```

ARMv8 Function Writing: Global Array

```
37                .text
38                .globl main
39 main:          str lr, [sp, #-16]!     //Adjusted stack pointer
40                                        //and save return address
41                ldr x0, =newline
42                bl printf
43
44                ldr x0, =source         //Display source array
45                mov x1, ARRAY_SIZE
46                bl show
47
48                ldr x0, =newline
49                bl printf
50
51                ldr x0, =destination    //Copy source array
52                ldr x1, =source         //into destination array
53                mov w2, ARRAY_SIZE
54                bl copy
55
56                ldr x0, =destination    //Display destination array
57                mov x1, ARRAY_SIZE
58                bl show
59
60                ldr x0, =newline
61                bl printf
62
63
64                ldr lr, [sp], #16       //Load return address and
65                                        //adjust stack pointer
66                mov x0, xzr             //Return zero to the caller
67                ret lr
68
```

ARMv8 Function Writing: Global Array

```
1               .data
2   elmfmt:     .asciz "%d "
3
4
5               .text
6               .global show
7
8   show:       sub sp, sp, #32          //Adjust stack pointer
9               str lr, [sp, #16]        //Save the return address
10              str x0, [sp]             //Save the array address
11              str w1, [sp, #8]         //Save the array size
12              mov w2, #0               //Initialize array index to 0
13              str w2, [sp, #12]        //Save the array index
14
15  loop:       ldr x0, =elmfmt          //Load the format onto x0
16              ldr x1, [sp]             //Load the saved array address
17              ldr w2, [sp, #12]        //Load the saved array index
18              ldr x1, [x1, x2, lsl#3] //Load the array element
19              add w2, w2, #1           //Increment array index
20              str w2, [sp, #12]        //Save incremented array index
21              bl printf                //Call printf
22
23              ldr w1, [sp, #8]         //Load saved array saved size
24              ldr w2, [sp, #12]        //Load the incremented arra index
25              cmp w2, w1               //Compare array index against array size
26              blt loop                 //Loop if the index is less than the size
27
28              ldr lr, [sp, #16]        //Load return address
29              add sp, sp, #32          //Adjust stack pointer
30              ret lr
31
```

ARMv8 Function Writing: Global Array

```
1
2               .text
3               .global copy
4
5   copy:       mov x3, #0                  //Initialize array index
6   loop:       ldr x4,[x1, x3, lsl#3]      //Load array element from source
7               str x4,[x0, x3, lsl#3]      //Save array element onto destination
8               add x3, x3, #1              //Incement array index
9               cmp x3,x2                   //Campre array index against its size
10              blt loop                    //Loop if the index is less than size
11              ret lr
12
```

ARMv8 Function Writing: Recursive

```c
#include <stdio.h>

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

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

```
1
2           .equ number, 5
3
4           .data
5   strfmt: .asciz "\nFactorial of %d is %d"
6
7           .text
8           .global main
9
10  main:   str lr, [sp, #-16]!
11          mov x0, number
12          bl factorial
13          mov x2, x0
14          mov x1, number
15          ldr x0, =strfmt
16          bl printf
17          ldr lr, [sp], #16
18          ret lr
19
```

Function or Subroutine Writing: Recursive

```
1         .text
2         .global factorial          // Making factorial function name visible
3                                    // to the codes in the other files
4 factorial:                         // Start of the factorial function
5         cmp x0, #1                 // Compare the function parameter with 1
6         ble base                   // Function parameter is less than or equal to 1,
7                                    // Jump to base case
8 recur:                             // Recursion part of factorial function
9         sub sp, sp, #16            // Adjust the stack pointer to push current lr,
10                                   // x0 register values onto the stack
11        stur lr, [sp, #8]          // Push the current lr register value on to the stack
12        stur x0, [sp, #0]          // Push the current x0, which holds the parameter value,
13                                   // onto the stack
14        sub x0, x0, #1             // Subtract 1 from the current parameter value n to pass
15                                   // it to the next recursive call of factorial(n-1) function
16        bl factorial               // Call factorial(n-1) function recursively
17                                   // Recursive call factorial(n-1) returned, with result in x0
18        ldur x1, [sp, #0]          // Pop the pushed parameter value n from the stack into x1
19        mul x0, x0, x1             // x0=n*factorial(n-1)
20        ldur lr, [sp, #8]          // Pop the pushed lr register value from the stack
21        add sp, sp, #16            // Adjust the stack pointer
22        ret lr                     // Return from the current call of factorial(n) frunction with
23                                   // the result in x0
24 base:                             // Base case of factorial function
25        mov x0, #1                 // Return 1 from the base case
26        ret lr                     // Return from factorial function
27
```

ARMv8 Function Writing: Using Pointer

```
char *string="NANAIMO";

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

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

ARMv8 Function Writing: Using Pointer

```
1           .data
2 string:   .asciz "*NANAIMO*"
3
4           .text
5           .global main
6
7 main:     str lr, [sp, #-16]!
8           ldr x0, =string
9           bl printf
10          ldr x0, =string
11          bl strlen
12          sub x1, x0, #1
13          ldr x0, =string
14          add x1, x0, x1
15          bl reverse
16          ldr x0, =string
17          bl printf
18          ldr lr, [sp], #16
19          ret lr
```

```
1               .text
2               .global reverse
3 reverse:      cmp x0, x1
4               bge done
5               ldrb w3, [x0]
6               ldrb w4, [x1]
7               strb w4, [x0]
8               strb w3, [x1]
9               add x0, x0, #1
10              sub x1, x1, #1
11              str lr, [sp, #-16]!
12              bl reverse
13              ldr lr, [sp], #16
14 done:        ret lr
```

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 main() {
        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         .equ struct_size, 80
2         .equ st_fn_offset, 0
3         .equ st_ln_offset, 30
4         .equ st_class_offset, 60
5         .equ st_grade_offset, 64
6
7
8         .data
9   fname:  .asciz "Humayun"
10  lname:  .asciz "Kabir"
11  strfmt: .asciz "\nStudent: [%s %s, %d, %d]\n"
12
```

```
13        .text
14        .global main
15  main:
16        str lr, [sp, #-16]!                 //Save return address and
17                                            //adjust stack pointer
18        sub sp, sp, struct_size             //Adjust stack pointer for
19                                            //local student structure
20        mov x0, sp                          //Load student address into x0
21        add x0, x0, st_fn_offset            //Add first name offset
22        ldr x1, =fname                      //Load the address of the string
23                                            //to copy into first name
24        bl strcpy                           //Call strcpy
25
26        mov x0, sp                          //Load student address into x0
27        add x0, x0, st_ln_offset            //Add last name offset
28        ldr x1, =lname                      //Load the address of the string
29                                            //to copy into last name
30        bl strcpy                           //Call strcpy
31
32        mov w8,#5                           //Initialize w8 with class value
33        strb w8, [sp, st_class_offset]      //Save class value at class offset
34        mov w8, #100                        //Initialize w8 with grade value
35        str w8, [sp, st_grade_offset]       //Save grade value at grade offset
36
37        ldr x0, =strfmt                     //Load student print format into x0
38        mov x1, sp                          //Load student address into x1
39        add x1, x1, st_fn_offset            //Add first name offset
40        mov x2, sp                          //Load student address into x2
41        add x2, x2, st_ln_offset            //Add last name offset
42        ldrb w3, [sp, st_class_offset]      //Load class value of student into w3
43        ldr w4, [sp, st_grade_offset]       //Load grade value of student into w4
44        bl printf                           //Call printf
45
46        mov w0, wzr
47        add sp, sp, struct_size             //Adjust stack pointer for student
48        ldr lr, [sp], #16                   //Load return address and adjust
49                                            //stack pointer
50        ret lr
```

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 main() {
        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 struct_size, 68
3           .equ st_fn_offset, 0
4           .equ st_ln_offset, 30
5           .equ st_class_offset, 60
6           .equ st_grade_offset, 64
7
8
9           .data
10 fname:    .asciz "Humayun"
11          .align 3, 0
12 lname:    .asciz "Kabir"
13          .align 3, 0
14 strfmt:   .asciz "\nStudent: [%s %s, %d, %d]\n"
15          .align 3, 0
16 student:  .skip struct_size, 0
17
```

```
18          .text
19          .global main
20 main:    str lr, [sp, #-16]!             //Save return address and adjust
21                                          //stack pointer
22          ldr x0, =student                //Load student address into x0
23          add x0, x0, st_fn_offset        //Add first name offset
24          ldr x1, =fname                  //Load the addres of the string
25                                          //to copy into fist name
26          bl strcpy                       //Call strcpy
27
28          ldr x0, =student                //Load student address into x0
29          add x0, x0, st_ln_offset        //Add last name offset
30          ldr x1, =lname                  //Load the address of the string
31                                          //to copy into last name
32          bl strcpy                       //Call strcpy
33
34          ldr x0, =student                //Load student address into x0
35          mov w8,#5                       //Initialize w8 with class value
36          strb w8, [x0, st_class_offset]  //Save class value at class offset
37          mov w8, #100                    //Initialize w8 with grade value
38          str w8, [x0, st_grade_offset]   //Save grade value at grade offset
39
40
41          ldr w4, [x0, st_grade_offset]   //Load student grade into w4
42          ldrb w3, [x0, st_class_offset]  //Load student class inti w3
43          mov x2, x0                      //Load student address into x2
44          add x2, x2, st_ln_offset        //Add last name offset
45          mov x1, x0                      //Load student address into x1
46          add x1, x1, st_fn_offset        //Add fist name offset
47          ldr x0, =strfmt                 //Load print format string address
48                                          //into x0
49          bl printf                       //Call printf
50
51          mov x0, xzr
52          ldr lr, [sp], #16               //Load return address and adjust
53                                          //stack pointer
54          ret lr
```

ARMv8 Scaled Address

```
1
2               .data
3   num1:       .word 10
4               .align 3, 0
5   num2:       .word -20
6               .align 3, 0
7   num3:       .word 30
8               .align 3, 0
9   num4:       .word -40
10              .align 3, 0
11  num5:       .word 50
12              .align 3, 0
13  strfmt:     .asciz "Current number: %d\n"
14
15
16              .text
17              .global main
18  main:       sub sp, sp, #16
19              str lr, [sp]
20
21              ldr x1, =num1
22              ldr x1, [x1]
23              ldr x0, =strfmt
24              bl printf
25              ldr x1, =num2
26              ldr x1, [x1]
27              ldr x0, =strfmt
28              bl printf
29              ldr x1, =num3
30              ldr x1, [x1]
31              ldr x0, =strfmt
32              bl printf
33              ldr x1, =num4
34              ldr x1, [x1]
35              ldr x0, =strfmt
36              bl printf
37              ldr x1, =num5
38              ldr x1, [x1]
39              ldr x0, =strfmt
40              bl printf
41
42              mov x0, xzr
43              ldr lr, [sp]
44              add sp, sp, #16
45              ret lr
46
```

ARMv8 Unscaled Address

```
1
2               .data
3   num1:       .word -10
4   num2:       .word 20
5   num3:       .word -30
6   num4:       .word 40
7   num5:       .word -50
8   strfmt:     .asciz "Current number: %d\n"
9
10
11              .text
12              .global main
13  main:       sub sp, sp, #16
14              str lr, [sp]
15
16              ldr x1, =num1
17              ldursw x1, [x1]
18              ldr w0, =strfmt
19              bl printf
20              ldr x1, =num2
21              ldursw x1, [x1]
22              ldr x0, =strfmt
23              bl printf
24              ldr x1, =num3
25              ldursw x1, [x1]
26              ldr x0, =strfmt
27              bl printf
28              ldr x1, =num4
29              ldursw x1, [x1]
30              ldr x0, =strfmt
31              bl printf
32              ldr x1, =num5
33              ldursw x1, [x1]
34              ldr x0, =strfmt
35              bl printf
36
37              mov x0, xzr
38              ldr lr, [sp]
39              add sp, sp, #16
40              ret lr
41
```