Functions (10A)

Young Won Lim
7/29/20

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

This document was produced by using LibreOffice.

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

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

Supporting Procedures

1. put parameters in a place where the procedure can access them
2. transfer control to the procedure
3. acquire the storage resources needed fr the procedure
4. perform the desired task
5. put the result value in a place where the calling program can access it
6. return control to the points of origin, since a procedure can be called
   from several points in a program

Computer Organization and Design ARM Edition: The Hardware Software Interface by D. A. Patterson and J. L. Hennessy

Registers

R0, R1, R2, R3 : four argument registers to pass parameters

LR : one link register containing the return address register
to the point of origin

Computer Organization and Design ARM Edition: The Hardware Software Interface by D. A. Patterson and J. L. Hennessy

Registers

```
BL ProcedureAddress
```

```
MOV     PC, LR
```

Computer Organization and Design ARM Edition: The Hardware Software Interface by D. A. Patterson and J. L. Hennessy

A procedure that does not call another procedures

```
int leaf_example (int g, int h, int I, int j)
{
        int f;
        f = (g + h) - (i+j);
        return f;
}
```

```
SUB     SP, SP, #12      ; adjust stack to make room for 3 items
STR     R6, [SP, #8]     ; save register R6 for a later use
STR     R6, [SP, #4]     ; save register R5 for a later use
STR     R6, [SP, #0]     ; save register R4 for a later use
```

Computer Organization and Design ARM Edition: The Hardware Software Interface by D. A. Patterson and J. L. Hennessy

A procedure that does not call another procedures

```
int leaf_example (int g, int h, int I, int j)
{
        int f;
        f = (g + h) - (i+j);
        return f;
}
```

```
ADD     R5, R0, R1          ; R5 = g + h
ADD     R6, R2, R3          ; R6 = I + j
SUB     R4, R5, R6          ; R4 = R5 - R6

MOV     R0, R4              ; returns f (R0 = R4)
```

Computer Organization and Design ARM Edition: The Hardware Software Interface by D. A. Patterson and J. L. Hennessy

A procedure that does not call another procedures

```
int leaf_example (int g, int h, int I, int j)
{
        int f;
        f = (g + h) - (i+j);
        return f;
}
```

```
LDR     R4, [SP, #0]        ; restore R4 for the caller
LDR     R5, [SP, #4]        ; restore R5 for the caller
LDR     R6, [SP, #8]        ; restore R6 for the caller
ADD     SP, SP, #12         ; adjust stack t delete 3 items


MOV     PC, LR              ; jump back to calling procedure
```

Computer Organization and Design ARM Edition: The Hardware Software Interface by D. A. Patterson and J. L. Hennessy

Instructions for procedures

```
BL   ProcedureAddress
```

jumps to an address and simultaneously saves
the address of the following instruction in register LR

```
MOV      PC, LR
```

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

Recursive procedure

```
Int fact (int n)
{
        if (n < 1) return (1);
        else return (n * fact(n-1));
}
```

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

Recursive procedure

```
Fact:
          SUB     SP, SP, #8          ; adjust stack for 2 items
          STR     LR, [SP, #8]        ; save the return address
          STR     R0, [SP, #0]        ; save the argument n

          CMP     R0, #1              ; compare n to 1
          BGE     L1                  ; if n >= 1, go to L1

          MOV     R0, #1              ; return 1
          ADD     SP, SP, #8          ; pop 2 items off stack
          MOV     PC, LR              ; return to the caller
```

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

Recursive procedure

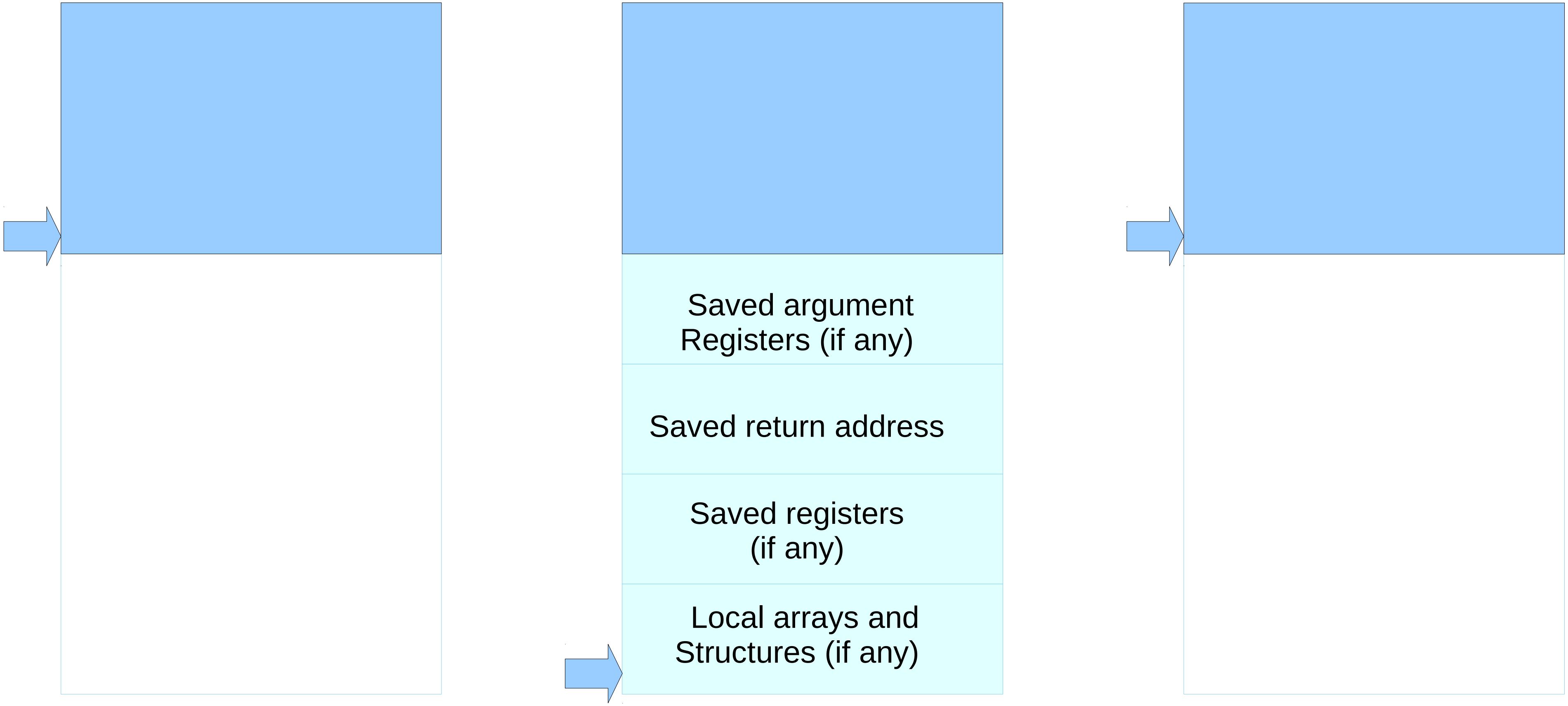
```
L1:
          SUB     R0, R0, #1            ; n >= 1 argument gets (n-1)
          BL      fact                  ; call fact with (n-1)

          MOV     R12, R0               ; save the return value
          LDR     R0, [SP, #0]          ; return from BL ; restore argument n
          LDR     LR, [SP, #0]          ; restore the return address
          ADD     SP, SP, #8            ; adjust stack pointer to pop 2 items

          MUL     R0, R0, R12           ; return n * fact (n-1)
          MOV     PC, LR                ; return to the caller
```

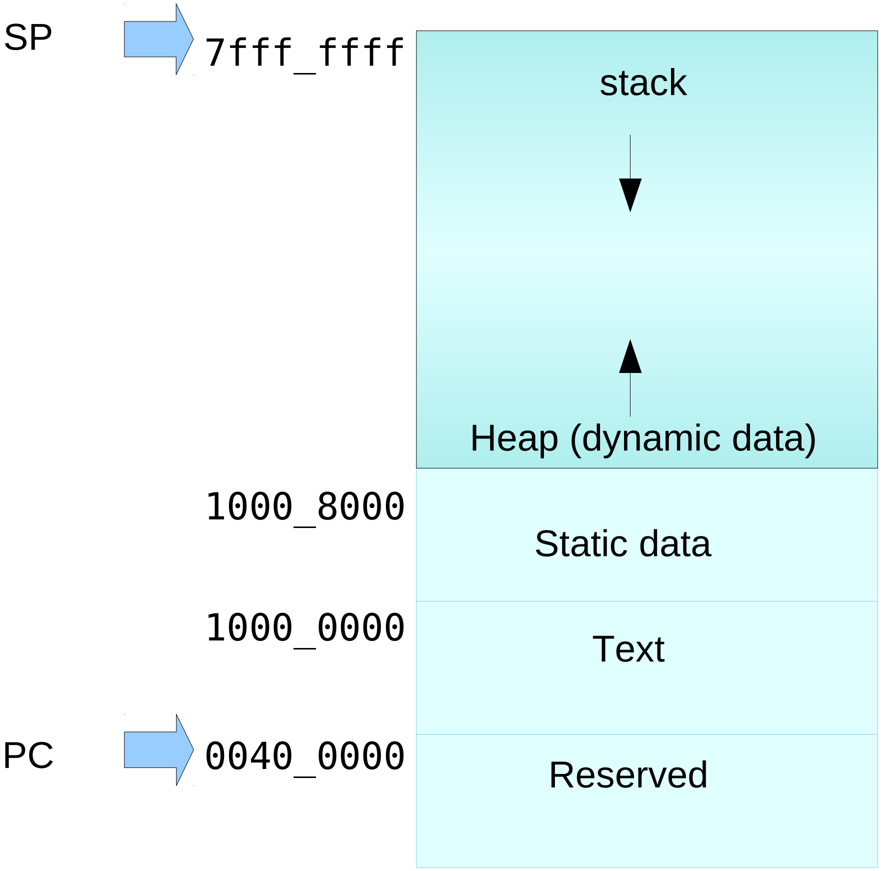
Introduction to ARM Cortex-M Microcontrollers – Embedded Systems, Jonathan W. Valvano

Stack allocation



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

Memory map



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

RM Register Conventions

Names	Reg No	Usage	preserved
a1-a2	0-1	Argument / return result/ scratch register	no
a3-a4	2-3	Argument / scratch register	no
v1-v8	4-11	Variables for local routine	yes
ip	12	Intra procedure call scratch register	no
sp	13	Stack pointer	yes
lr	14	Link register (Return address)	yes
pc	15	Program counter	n.a.

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

Recursive Procedure and Iterative Implementation

```
int sum (int n, int acc) {
        if (n > 0)
                return sum(n-1, acc+n);
        else
                return acc;
}

Sum:    CMP     R0, #0              ; test if n <= 0
        BLE sum_exit                ; go to sum_exit if n <= 0;
        ADD     R1, R1, R0          ; add n to acc
        SUB     R0, R0, #1          ; subtract 1 from n
        B     sum                   ; go to sum
sum_exit:
        MOV     R0, R1              ; return value acc
        MOV     PC, LR              ; return to caller
```

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

String Copy Procedure

```
void strcpy (char x[], char y[])
{
        int i;

        i = 0;
        while ((x[i] = y[i]) != ‘\0’)              // copy & test byte
                i += 1;
}
```

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

String Copy Procedure

```
Strcpy:  SUB    SP, SP, #4       ; adjust stack for 1 more item
         STR    R4, [SP, #0]     ; save R4
         MOV    R4, #0           ; I = 0 + 0
L1:      ADD    R2, R4, R1       ; address of y[i] in R2
         LDRBS  R3, [R2, #0]     ; R3 = y[i] and set condition flag
         ADD    R12, R4, R0      ; addressof x[i] in r12
         STRB   R3, [R12, #0]    ; x[i] = y[i]
         BEQ    L2               ; if y[i] == 0, go to L2
         ADD    R4, R4, #1       ; I = i+1
         B      L1               ; go to L1
L2       LDR    R4, [SP, #0]     ; y[i] == 0 : end of string, restore old R4
         ADD    SP, SP, #4       ; pop 1 word off stack
         MOV    PC, LR           ; return
```

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

Sort

```
void swap(int v[], int k)
{
        int temp;
        temp = v[k];
        v[k] = v[k+1];
        v[k+1] = temp;
}
```

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

Sort

```
v        RN  0    ;  1st argument address of y
k        RN  1    ; 2nd argument index k
temp     RN  2    ; local variable
temp2    RN  3    ; temporary for v[k+1]
vkAddr   RN  12   ; to hold address of v[k]
```

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

String Copy Procedure

```
swap:   ADD     vkAddr, v, k, LSL #2        ; reg vkAddr = v + (k * 4)
                                            ; reg vkAddr has the address of v[k]
        LDR     temp, [vkAddr, #0]          ; temp = v[k]
        LDR     temp2, [vkAddr, #4]         ; temp2 = v[k+1]
                                            ; refers to next element of v
        STR     temp2, [vkAddr, #0]         ; v[k] = temp2
        STR     temp, [vkAddr, #4]          ; v[k+1] = temp

        MOV     PC, LR                      ; return to calling routine
```

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

Instructions for procedures

```
B{cond}    label   ; branch to label
BX{cond}   Rm      ; branch indirect to location specified by Rm
BL{cond}   label   ; branch to subroutine at label
BLX{cond}  Rm      ; branch to subroutine indirect specified by Rm
```

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

Instructions for procedures

```
uint32_t Num;

void Change(void) {
        Num = Num + 25;
}

void main(void) {
        Num = 0;
        while (1) {
                Change();
        }
}
```

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

Instructions for procedures

```
Change  LDR     R1, =Num        ; 5) R1 = &Num
        LDR     R0, [R1]        ; 6) R0 = Num
        ADD     R0, R0, #25     ; 7) R0 = Num + 25
        STR     R0, [R1]        ; 8) Num = Num + 25
        BX      LR              ; 9) return

Main    LDR     R1, =Num        ; 1) R1 = &Num
        MOV     R0, #0          ; 2) R0 = 0
        STR     R0, [R1]        ; 3) Num = 0
Loop    BL      Change          ; 4) call to Change
        B       Loop            ; 10) repeat
```

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

Instructions for procedures

```
uint32_t Num;

void Change(void) {
        if (Num <25600) {
                Num = Num + 25;
        }
}

void main(void) {
        Num = 0;
        while (1) {
                Change();
        }
}
```

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

Instructions for procedures

```
Change  LDR     R1, =Num        ; R1 = &Num
        LDR     R0, [R1]        ; R0 = Num
        CMP     R0, #25600      ;
        BHS     skip
        ADD     R0, R0, #25     ; R0 = Num + 25
        STR     R0, [R1]        ; Num = Num + 25
Skip    BX      LR              ; return

Main    LDR     R1, =Num        ; R1 = &Num
        MOV     R0, #0          ; R0 = 0
        STR     R0, [R1]        ; Num = 0
Loop    BL      Change          ; call to Change
        B       Loop            ; repeat
```

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

Instructions for procedures

```
uint32_t Num;

void Change(void) {
        if (Num <100) {
                Num = Num + 1;
        } else {
                Num = -100;
        }
}

void main(void) {
        Num = 0;
        while (1) {
                Change();
        }
}
```

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

Instructions for procedures

```
Change  LDR     R1, =Num        ; R1 = &Num
        LDR     R0, [R1]        ; R0 = Num
        CMP     R0, #100        ;
        BGE     else
        ADD     R0, R0, #1      ; R0 = Num + 1
        B       skip
Else    MOV     R0, #-100       ; R0 = -100
skip    STR     R0, [R1]        ; Num = Num + 1 or -100
        BX      LR              ; return

Main    LDR     R1, =Num        ; R1 = &Num
        MOV     R0, #0          ; R0 = 0
        STR     R0, [R1]        ; Num = 0
Loop    BL      Change          ; call to Change
        B       Loop            ; repeat
```

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

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