Chapter 2

Instructions: Language of the Computer

Tutorial 3

- Function calling
- Function calling examples
- Memory layout

Function Calling

- Steps required
  1. Place parameters in registers
  2. Transfer control to callee function
  3. Acquire stack storage
  4. Back up registers
  5. Perform function's operations
  6. Place result in register for caller
  7. Restore registers
  8. Return to place of call

ARM register conventions

Name	Register number	Usage	Preserved on call?
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.

Function Call Instructions

- Fucntion call: Branch and link

```
BL Function_Address
```

  - Address of the following instruction put in ***lr***
  - Jumps to target address

- Return from function:

```
MOV pc,lr
```

  - Copies ***lr*** to program counter

Function Example

- C code:

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

  - *Arguments g, h, i, j in r0, r1, r2, r3*
  - *Put f in r4? Then need to save original value of r4 on stack. (True for registers r4-r11)*
  - *Result in r0*

Function Example

- ARM code:

```
example:
    SUB sp, sp, #12
    STR r6,[sp,#8]
    STR r5,[sp,#4]
    STR r4,[sp,#0]
    ADD r5,r0,r1
    ADD r6,r2,r3
    SUB r4,r5,r6
    MOV r0,r4
    LDR r4, [sp,#0]
    LDR r5, [sp,#4]
    LDR r6, [sp,#8]
    ADD sp,sp,#12
    MOV pc,lr
```

Make room for 3 items

Save r4,r5,r6 on stack

r5 = (g+h), r6= (i+j)
Result f in r4

Result moved to return value register r0.

Restore r4,r5,r6

Return

Exercise 6

- In ex6.s write the following function that computes the factorial of a given number using the iterative method.
  - int fact(int number)

Show your work to an instructor

Non-leaf Functions

- Functions that call other functions
- To support nested calls, the *caller function* needs to save on the stack:
  - Return address (lr) to the **previous caller**
  - Any arguments and temporary register values that will be needed after the nested call
- Restore from the stack after the nested call returns

Nested Calls Example

- C code:

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

  - *Argument n in register r0*
  - *Result in register r0*

Nested Calls Example

- ARM code:

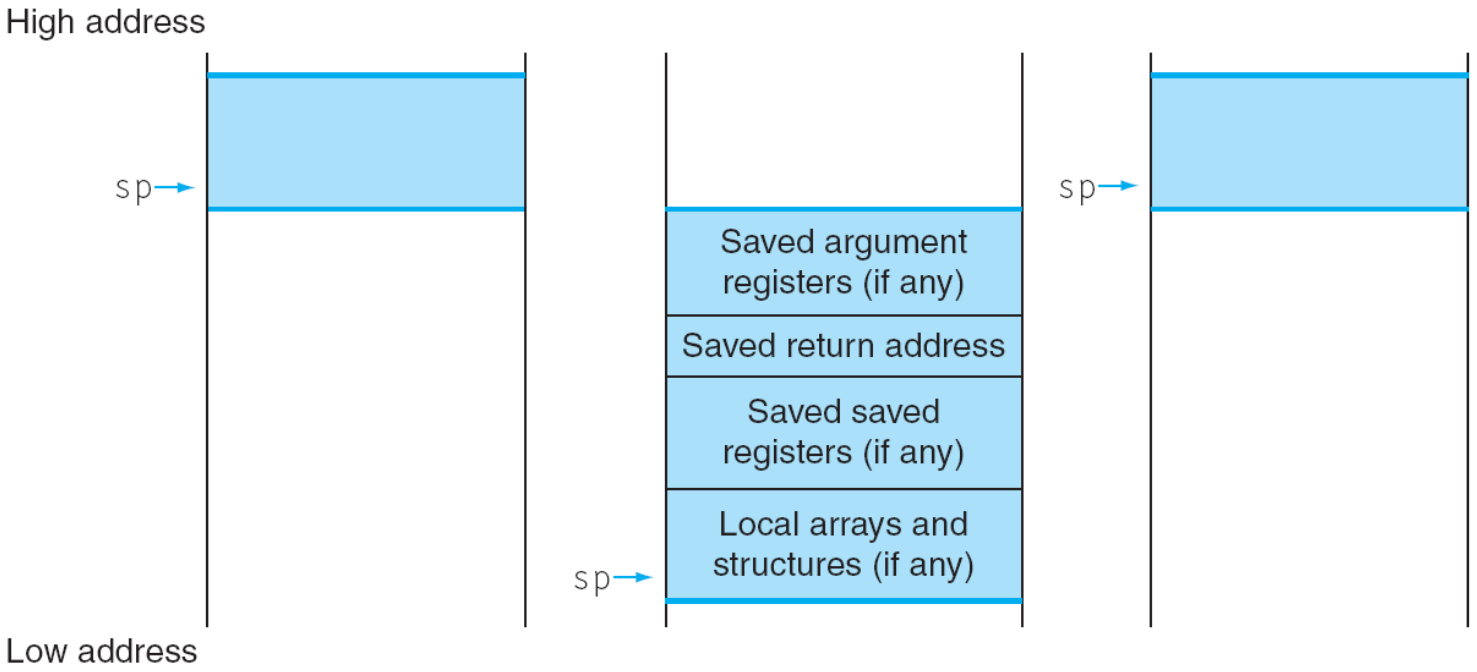
```
fact:
    SUB sp,sp,#8        ; Adjust stack for 2 items
    STR lr,[sp,#4]      ; Save return address
    STR r0,[sp,#0]      ; Save argument n
    CMP r0,#1           ; Compare n to 1
    BGE L1
    MOV r0,#1           ; If so, result is 1
    ADD sp,sp,#8        ; Pop 2 items from stack
    MOV pc,lr           ; Return to caller
L1: SUB r0,r0,#1        ; Else decrement n
    BL  fact            ; Nested call
    MOV r12,r0          ; Obtain returned value
    LDR r0,[sp,#0]      ; Restore original n
    LDR lr,[sp,#4]      ; Restore return address
    ADD sp,sp #8        ; pop 2 items from stack
    MUL r0,r0,r12       ; Multiply to get result
    MOV pc,lr           ; Return
```

Exercise 7

- In ex7.s write the following function that computes Fibonacci numbers using recursion.
  - int Fibonacci(int number)
  - Fibonacci sequence :
    - $F_n=F_{n-1}+F_{n-2}$
    - Where $F_1=1$ and $F_2=1$

Show your work to an instructor

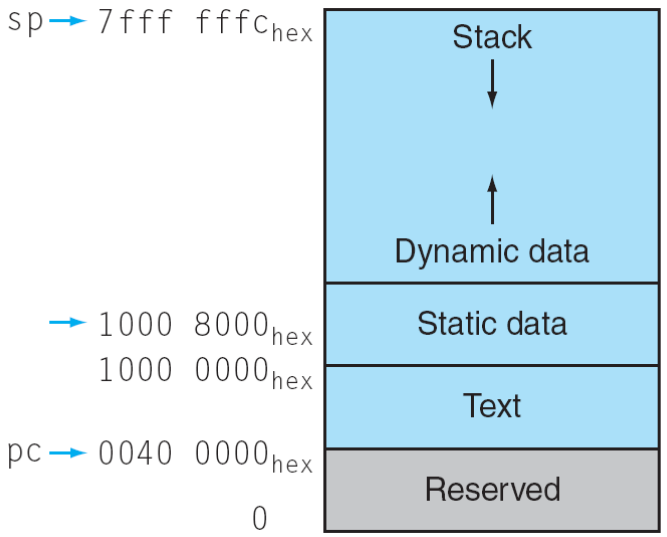
Local Data on the Stack



- Local data allocated by callee
  - e.g., C automatic variables
- Stack frame (activation record)
  - Segment of the stack containing a function's saved registers and local variables

Memory Layout

- Text: program code
- Static data:
  - e.g., global variables, static variables in C and strings
- Dynamic data: heap
  - E.g., malloc in C, new in Java
- Stack: automatic storage



C Sort Example

- Illustrates use of assembly instructions for a C bubble sort function
- First, define Swap function (a leaf function – no nested call)

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

The Swap Function

Assembler directive

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

Procedure body

```
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 (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
```

Procedure return

```
       MOV    pc, lr                ; return to calling routine
```

The Sort function in C

- A non-leaf function (calls swap)

```
void sort (int v[], int n)
{
  int i, j;
  for (i = 0; i < n; i += 1) {
    for (j = i - 1;
         j >= 0 && v[j] > v[j + 1];
         j -= 1) {
      swap(v,j);
    }
  }
}
```

Register allocation and saving registers for *sort*

Register allocation

v	RN 0	; 1st argument address of v
n	RN 1	; 2nd argument index n
i	RN 2	; local variable i
j	RN 3	; local variable j
vjAddr	RN 12	; to hold address of v[j]
vj	RN 4	; to hold a copy of v[j]
vj1	RN 5	; to hold a copy of v[j+1]
vcopy	RN 6	; to hold a copy of v
ncopy	RN 7	; to hold a copy of n

Saving registers

```
sort:   SUB   sp,sp,#20          ; make room on stack for 5 registers
        STR   lr, [sp, #16]      ; save lr on stack
        STR   ncopy, [sp, #12]   ; save ncopy on stack
        STR   vcopy,  [sp, #8]   ; save vcopy on stack
        STR   j,  [sp, #4]       ; save j on stack
        STR   i,  [sp, #0]       ; save i on stack
```

Function body - *sort*

Move parameters		MOV	vcopy, v	; copy parameter v into vcopy (save r0)
		MOV	ncopy, n	; copy parameter n into ncopy (save r1)
Outer loop		MOV	i, #0	; i = 0
	for1tst:	CMP	i, n	; if i ≥ n
		BGE	exit1	; go to exit1 if i ≥ n
Inner loop		SUB	j, i, #1	; j = i - 1
	for2tst:	CMP	j, #0	; if j < 0
		BLT	exit2 ; go to exit2 if j < 0	
		ADD	vjAddr, v, j, LSL #2	; reg vjAddr = v + (j * 4)
		LDR	vj, [vjAddr,#0]	; reg vj = v[j]
		LDR	vj1, [vjAddr,#4]	; reg vj1 = v[j + 1]
		CMP	vj, vj1	; if vj ≤ vj1
		BLE	exit2	; go to exit2 if vj ≤ vj1
Pass parameters and call		MOV	r0, vcopy	; first swap parameter is v
		MOV	r1, j	; second swap parameter is j
		BL	swap	; swap code shown in Figure 2.23
Inner loop		SUB	j, j, #1	; j -= 1
		B	for2tst	; branch to test of inner loop
Outer loop	exit2:	ADD	i, i, #1	; i += 1
		B	for1tst	; branch to test of outer loop

Restoring registers and return - *sort*

```
exit1:   LDR    i,  [sp, #0]          ; restore i from stack
         LDR    j,  [sp, #4]          ; restore j from stack
         LDR    vcopy,  [sp, #8]      ; restore vcopy from stack
         LDR    ncopy, [sp, #12]      ; restore ncopy from stack
         LDR    lr, [sp, #16]         ; restore lr from stack
         ADD    sp,sp,#20             ; restore stack pointer
```

Procedure return

```
         MOV    pc, lr                ; return to calling routine
```