CS2208A
TUTORIAL #8

November 25, 2013

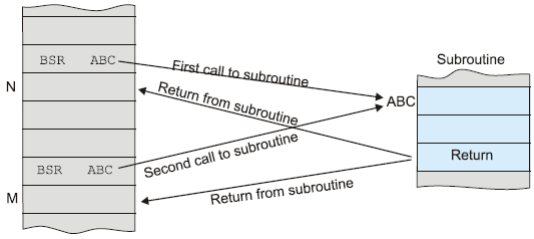
Overview

- Multiple Subroutine Calls
- Block Move Instructions
- Passing Parameters to a Subroutine
- Improving the Code

Multiple Subroutine Calls

3

- Subroutine return addresses are stacked.
- Can call subroutines from a subroutine (nesting).
- Done with ***block move instructions***.
- Copy multiple registers to and from the stack.



Block Move Instructions

4

- Instructions that perform multiple actions.
- Copy several registers to or from memory.
  - **STMFD:** Push a group of registers on the stack
  - **LDMFD:** Pull a group of registers off the stack
- STMFD *store multiple registers full descending*.
  - The term ***full*** means that the stack points at the top item on the stack
  - The term ***descending*** tells you that the stack grows towards lower addresses as items are pushed on top

Block Move Instructions

5

- Store data on the stack with SP (r13).
- Write SP! to use automatic indexing.
- Enclose register list between braces.
  - {r0,r1,r7} specifies registers r0, r1 and r7
- Use a dash to denote a sequence of registers.
  - {r0-r5,r8,r11} indicates the register list r0, r1, r2, r3, r4, r5, r8, and r11

Block Move Instructions

6

- To push r0 and r1 on the stack:
  - STMFD sp!,{r0,r1}
- To pull r0 and r1 off the stack:
  - LDMFD sp!,{r0,r1}
- Let's see an example (pg.57 of the workbook).

Block Move Instructions

7

```
      AREA  BlockMove,CODE,READWRITE   ;make readwrite because we have the stack in this area
      ADR   sp,Base                   ;point to the base of the stack
      MOV   r0,#0xAB                  ;dummy value for r0
      MOV   r1,#0xCD                  ;dummy value for r1
      MOV   lr,#0xDE                  ;dummy value for link register, r14
      BL    SQR1                      ;call Test
Loop  B     Loop                      ;stay here

Test  STMFD sp!,{r0,r1,lr}            ;save r0, r1, lr on the stack
      MOV   r0,#0x11                  ;let's do something pointless
      MOV   r1,#0x22                  ;let's do something pointless
      MOV   r14,#0x22                 ;let's change the link register
      ADD   r3,r0,r1                  ;ladd r0 and  r1 and put the result in r3
      LDMFD sp!,{r0,r1,pc}            ;pull r0, r1, lr off the stack

      DCD   0x89ABCDEF,0,0,0,0x12345678 ;stack area
Base  DCD   0xAAAAAAAA                ;stack base and dummy data
      END
```

Correct it to Test, not SQR1

Passing Parameters to a Subroutine

8

- We often have to pass parameters to subroutines.
- High-level language:
  - XYZ(P,Q).
- Low-level language:
  - Push the parameters on the stack
  - Push immediately before calling the subroutine
- You don't have to pass parameters via the stack.
- You can transfer them with the registers.

Passing Parameters to a Subroutine

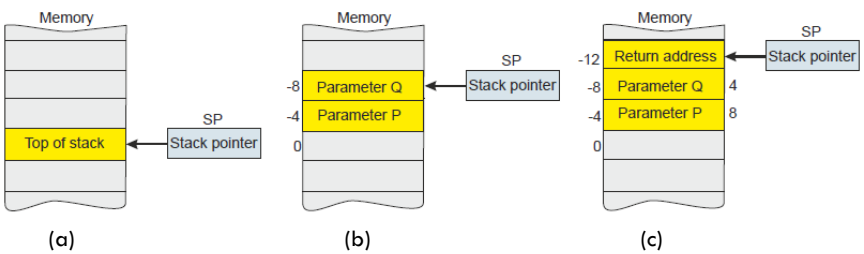
9

- Lets call a subroutine that adds two numbers P and Q and returns their result S = P + Q.
- Pseudocode:
  - Push P
  - Push Q
  - Call ADD
  - Pull S
  - Adjust the stack

Passing Parameters to a Subroutine

10

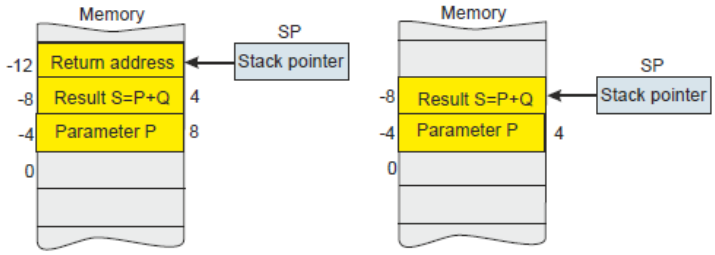
- Push the parameters on the stack and call the subroutine
  - (a) Immediately before the subroutine is called.
  - (b) Both parameters have been pushed.
  - (c) Subroutine has been called and the return address is saved on the stack.



Passing Parameters to a Subroutine

11

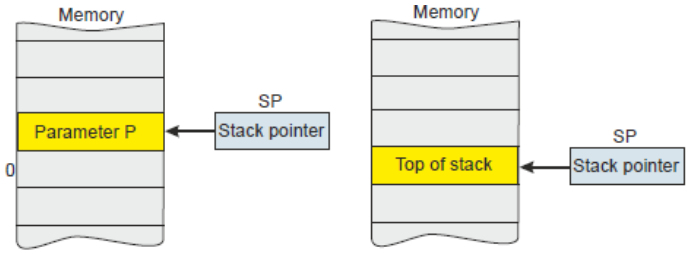
- The two parameters are pulled off the stack.
- The result replaces one of the parameters.



Passing Parameters to a Subroutine

12

- We have to adjust the stack since we have pushed two parameters but pulled only one.
- The stack must always be balanced with an equal numbers of push and pull operations.



Passing Parameters to a Subroutine

13

- The stack grows as parameters are pushed and the subroutine called.
- The stack declines as a return made and the two items on the stack removed.
- Let's see an example (pg.61 of the workbook).

Passing Parameters to a Subroutine

14

```
      AREA  ParamTest,CODE,READWRITE   ; make readwrite because of the stack
      ADR   sp,Base                    ; point to the base of the stack
      MOV   r0,#0xAB                   ; dummy value for P in r0
      MOV   r1,#0xCD                   ; dummy value for Q in r1
      STR   r0,[sp,#-4]!               ; push P
      STR   r1,[sp,#-4]!               ; push Q
      BL    ADDR                       ; call the adder
      LDR   r2,[sp],#4                 ; pull S off the stack
      ADD   sp,sp,#4                   ; adjust the stack pointer
Loop  B     Loop                       ; park here

ADDR  STR   lr,[sp,#-4]!               ; push the link register on the stack
      LDR   r5,[sp,#8]                 ; get P (buried under the return address and Q)
      LDR   r6,[sp,#4]                 ; get q (buried under the return address)
      ADD   r5,r5,r6                   ; do the addition
      STR   r5,[sp,#4]                 ; save result on the stack under return address (overwrite Q)
      LDR   pc,[sp],#4                 ; pull return address off the stack

      DCD   0,0,0,0,0                  ; stack area
Base  DCD   0xAAAAAAAA                 ; stack base and dummy data as marker
      END
```

Improving the Code

15

- The previous example is not efficient coding.
- Use block move instructions instead.
- Let's see an example (pg.67 of the workbook).

Improving the Code

16

```
      AREA   ParamTest1,CODE,READWRITE ; make readwrite because we locate the stack in this area
      ADR    sp,Base                    ; point to the base of the stack
      MOV    r0,#0xAB                   ; dummy value for P in r0
      MOV    r1,#0xCD                   ; dummy value for Q in r1
      STMFD  sp!,{r0,r1}                ; push P and Q
      BL     ADDR                       ; call the addition subroutine
      LDMFD  sp!,{r0,r2}                ; pull S and P off the stack
Loop  B      Loop                       ; park here

ADDR  STMFD  sp!,{r5,r6,lr}             ; push the link register and working registers
      LDR    r5,[sp,#16]                ; get P (buried under the return address and Q)
      LDR    r6,[sp,#12]                ; get Q (buried under the return address)
      ADD    r5,r5,r6                   ; do the addition
      STR    r5,[sp,#16]                ; save result on the stack under the return address
      LDMFD  sp!,{r5,r6,pc}             ; pull return address and working registers

      DCD   0xFFFFFFFF,0,0,0,0,0        ; stack area
Base  DCD   0xAAAAAAAA                  ; stack base and dummy data
      END
```