The Hardware Stack

CSE 30: Computer Organization and Systems Programming

Diba Mirza
Dept. of Computer Science and Engineering
University of California, San Diego
All about Stacks

- Follow the last in first out (LIFO) principle
  - Operations: push, pop
- Software stacks: Program with a LIFO interface
  Implementations are based on:
  - Arrays
  - Linked-lists
- Hardware stacks: Physical memory with LIFO access
  - Fixed starting location, the memory area is variable length
  - Each entry is a fixed size
  - A register points to the top of the stack
Hardware Stack Models

In general the register ‘sp’ holds the address of the ‘top’ of the stack
1. Ascending/Descending

2. Full/Empty
The AAPCS specifies a
- Full descending stack
- Stack width is 8 bytes
**int i = 10;**

- Which of the following ARM statements stores the local variable i on the stack according to the ARM procedure call standard?

  A. MOV r0, #10
     SUB sp, sp, #8
     LDR r0, [sp]

  B. MOV r0, #10
     SUB sp, sp, #8
     STR r0, [sp]

  C. MOV r0, #10
     SUB sp, sp, #4
     LDR r0, [sp]

  D. MOV r0, #10
     SUB sp, sp, #4
     STR r0, [sp]
Which of the following ARM statements stores the local variables $i$ and $j$ on the stack according to the ARM procedure call standard?

A. `MOV r0, #10
   MOV r1, #20
   SUB sp, sp, #8
   STR r0, [sp]
   STR r1, [sp, #4]`

B. `MOV r0, #10
   MOV r1, #20
   SUB sp, sp, #8
   STR r0, [sp]
   SUB r1, [sp]
   STR r1, [sp]`

C. Both are correct
New instruction STM

STMDB sp!, {r0 – r1}

MOV r0, #10
MOV r1, #20
SUB sp, sp, #8
STR r0, [sp]
STR r1, [sp, #4]
New instruction STM

STMDB sp!, \{r0 – r1\}

Are the two given ARM codes equivalent?
A. Yes
B. No

foo:
  MOV r0, #10
  MOV r1, #20
  SUB sp, sp, #8
  STR r0, [sp]
  STR r1, [sp, #4]
  BX lr

foo:
  MOV r0, #10
  MOV r1, #20
  STMDB sp!, \{r0, r1\}
  BX lr
C local variables

Is the given C and ARM code equivalent?

A. Yes
B. No

```c
int foo() {
    int i=10, j=20;
    return i+j;
}
```

```assembly
foo:
    MOV r0, #10
    MOV r1, #20
    STMDB sp!, {r0, r1}
    ADD r0, r0, r1
    BX lr
```
int read_and_sum() {
    int l1, l2;
    scanf("%d%d", &l1, &l2);
    return l1 + l2;
}

read_int:
    push {r4-r11, ip, lr}
    sub sp, sp, #8          @ reserve 8 bytes for the 2 local integers

    ldr r0, =scan_ints       @ format string
    mov r1, sp               @ address of l1
    add, r2, sp, #4          @ address of l2
    bl scanf

    ldr r0, [sp]             @ read l1
    ldr r1, [sp, #4]         @ read l2
    add r0, r0, r1

    add sp, sp, #8           @ de-allocate 8 bytes, restore sp to
    @ value after push

    pop {r4-r11, ip, lr}
    bx lr