PA4
The Hardware Stack

CSE 30: Computer Organization and Systems Programming

Diba Mirza
Dept. of Computer Science and Engineering
University of California, San Diego
Implement: is_substring

/* Check if s1 is a substring of s2 */
/* Return 1 if s1 is a substring of s2 */
/* Return 0 otherwise */

int is_substring (char *s1, char *s2){
    int i=0, j=0;
    if (s1==null || s2==null)
        return 0;
    if (s1[0]==0)
        return 1;
    while (s2[i]!=0){
        if (s2[i]==s1[0]){
            j=0;
            while (s1[j]!=0 && s2[i+j]!=0 && s1[j]==s2[i+j])
                j++;
            if (s1[j]==0)
                return 1;
        }
        i++;
    }
    return 0;
}
Stacks

- A stack follows 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: Area of memory that grows and shrinks according to the LIFO principle
  - Pointers define stack limits:
    - Base pointer: points to the “bottom” of the stack
    - Stack pointer: points to the “top” of the stack
  - Fixed width
Hardware Stack Models

- There are a few different models for implementing hardware stacks, different architectures follow different models

- ARM follows a Full Descending Model

- Why do I need to know these models?
  - Creating local variables
  - Understanding weird program behavior
  - Implement your own!
In general the register ‘sp’ holds the address of the ‘top’ of the stack

1. Ascending/Descending

- **Ascending**: Stack grows in the direction of increasing memory locations.
- **Descending**: Stack grows in the direction of decreasing memory locations.

2. Full/Empty

- **Full**: Stack pointer points to the last entry on the stack.
- **Empty**: Stack pointer points to the next available (free) spot on the stack.
Match the diagram to stack model

<table>
<thead>
<tr>
<th></th>
<th>(1)</th>
<th>(2)</th>
<th>(3)</th>
<th>(4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>FA</td>
<td>EA</td>
<td>FD</td>
<td>ED</td>
</tr>
<tr>
<td>B</td>
<td>FD</td>
<td>ED</td>
<td>FA</td>
<td>EA</td>
</tr>
<tr>
<td>C</td>
<td>ED</td>
<td>FD</td>
<td>EA</td>
<td>FA</td>
</tr>
<tr>
<td>D</td>
<td>EA</td>
<td>FA</td>
<td>ED</td>
<td>FD</td>
</tr>
</tbody>
</table>

- **FA**: Full Ascending
- **FD**: Full Descending
- **EA**: Empty Ascending
- **ED**: Empty Descending

```
0x418
```

```
0x400
```

```
0x3fca
```

```
0x3e8
```
ARM Procedure call standard

The AAPCS specifies a

- Full descending stack
- Stack width is 8 bytes

```
push  $r0, r1, r3-r5
LDR   $e6, [sp, #12]  // Read the value of r4 in stack
```

* sp should always be 8byte aligned

```
0x418
0x400
0x3e8
```

UCSD
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]  -