From last time

Explain what the ARM instruction BL does, and how control is returned to the calling code when the method finishes.

BL fred

then

B fred

(i.e. PC=fred)

return at end:

MOV PC, LR

If the method itself calls another method, how does this affect the return? (4 marks)

Why is extra memory (i.e. a stack) useful in the implementation of method calls?
From last time – ctd.

Why do we use a stack rather than fixed memory locations? (4 marks)

A stack is implemented using SP and pre-/post-indexing instructions. Give code for the ’push’ and ’pop’ operations and explain the value in SP after each operation. (4 marks)
COMP15111: Introduction to Architecture
Lecture 10: Methods – Parameters and Variables

Dr. Javier Navaridas

School of Computer Science, University of Manchester

Autumn 2016
Overview & Learning Outcomes

Using the stack for:

– Method Parameters (Arguments)
– Method Variables
– Saving Registers

Static Variables

More ARM instructions
Problems

Last lecture, we said:
e.g. R0 = 1st parameter, R1 = 2nd, R2 = 3rd etc.

What if we have a lot of parameters?

Efficiency: when one method calls others?

Efficiency: is this the best use of registers?

Alternative: push arguments onto stack during call
Complete (simple) example call again

\[\text{myage} = \text{age (myday, mymonth, myyear)};\]

LDR R0, myday
STR R0, [SP,#-4]!
LDR R0, mymonth
STR R0, [SP,#-4]!
LDR R0, myyear
STR R0, [SP,#-4]!
BL age
STR R0, myage
How does a method access its stacked parameters?

```c
int age (int day, int month, int year) {
    ...
}
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>year (myyear)</td>
<td>SP</td>
</tr>
<tr>
<td>month (mymonth)</td>
<td>SP+4</td>
</tr>
<tr>
<td>day (myday)</td>
<td>SP+8</td>
</tr>
</tbody>
</table>

```assembly
age
   ...; use [SP, #8] or [SP, #4] or [SP]
   ADD SP, SP, #12  ; discard 3 parameters
   MOV PC, LR       ; return
```
What if we also have to stack LR?

<table>
<thead>
<tr>
<th>year</th>
<th>← SP</th>
</tr>
</thead>
<tbody>
<tr>
<td>month</td>
<td>SP+4</td>
</tr>
<tr>
<td>day</td>
<td>SP+8</td>
</tr>
</tbody>
</table>

age STR LR, [SP, #-4]!

<table>
<thead>
<tr>
<th>LR</th>
<th>← SP</th>
</tr>
</thead>
<tbody>
<tr>
<td>year</td>
<td>SP+4</td>
</tr>
<tr>
<td>month</td>
<td>SP+8</td>
</tr>
<tr>
<td>day</td>
<td>SP+12</td>
</tr>
</tbody>
</table>

...; use [SP, #12] or [SP, #8] or [SP, #4]
LDR PC, [SP], #16; return and discard 3 parameters; and LR.
Local (method) variables

Most variables declared inside a method have the same lifetime as the method itself:
– when the method is called, its variables are created
– when the method returns, its variables are destroyed
(there are exceptions, but we will ignore them for now)

Animation: Lifespan of Variables

The most space-efficient way to do this is:
– at the method start, “push” extra space for its variables
– at the method end, “pop” the extra space

```
SUB SP, SP, #bytes of variables; create space
...
ADD SP, SP, #bytes of variables; destroy space
```
Example method call – save return link

```java
void printAgeHistory (int birthDay,
    int birthMonth, int birthYear) {  
    int someYear = birthYear + 1;
    int ageInSomeYear = 1;
    ...
}
```

at the start of `printAgeHistory`:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>birthYear</td>
<td>← SP</td>
</tr>
<tr>
<td>birthMonth</td>
<td>SP+4</td>
</tr>
<tr>
<td>birthDay</td>
<td>SP+8</td>
</tr>
</tbody>
</table>
Example – save return link

```
STR LR, [SP, #-4]! ; save return link
```

```
<table>
<thead>
<tr>
<th></th>
<th></th>
<th>← SP</th>
</tr>
</thead>
<tbody>
<tr>
<td>birthYear</td>
<td></td>
<td>SP+4</td>
</tr>
<tr>
<td>birthMonth</td>
<td></td>
<td>SP+8</td>
</tr>
<tr>
<td>birthDay</td>
<td></td>
<td>SP+12</td>
</tr>
</tbody>
</table>
```
Example – create space for variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>LR</td>
<td>← SP</td>
</tr>
<tr>
<td>birthYear</td>
<td>SP+4</td>
</tr>
<tr>
<td>birthMonth</td>
<td>SP+8</td>
</tr>
<tr>
<td>birthDay</td>
<td>SP+12</td>
</tr>
</tbody>
</table>

SUB SP, SP, #8 ; create space for 2 variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ageInSomeYear</td>
<td>← SP</td>
</tr>
<tr>
<td>someYear</td>
<td>SP+4</td>
</tr>
<tr>
<td>LR</td>
<td>SP+8</td>
</tr>
<tr>
<td>birthYear</td>
<td>SP+12</td>
</tr>
<tr>
<td>birthMonth</td>
<td>SP+16</td>
</tr>
<tr>
<td>birthDay</td>
<td>SP+20</td>
</tr>
</tbody>
</table>
Example – body of method

<table>
<thead>
<tr>
<th>ageInSomeYear</th>
<th>← SP</th>
</tr>
</thead>
<tbody>
<tr>
<td>someYear</td>
<td>SP+4</td>
</tr>
<tr>
<td>LR</td>
<td>SP+8</td>
</tr>
<tr>
<td>birthYear</td>
<td>SP+12</td>
</tr>
<tr>
<td>birthMonth</td>
<td>SP+16</td>
</tr>
<tr>
<td>birthDay</td>
<td>SP+20</td>
</tr>
</tbody>
</table>

LDR R0, [SP,#12] ; someYear = birthYear + 1
ADD R0, R0, #1
STR R0, [SP,#4]
MOV R0, #1 ; ageInSomeYear = 1
STR R0, [SP]
...
Example – remove space for variables

```
<table>
<thead>
<tr>
<th>Variable</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ageInSomeYear</td>
<td>SP</td>
</tr>
<tr>
<td>someYear</td>
<td>SP+4</td>
</tr>
<tr>
<td>LR</td>
<td>SP+8</td>
</tr>
<tr>
<td>birthYear</td>
<td>SP+12</td>
</tr>
<tr>
<td>birthMonth</td>
<td>SP+16</td>
</tr>
<tr>
<td>birthDay</td>
<td>SP+20</td>
</tr>
</tbody>
</table>
```

ADD SP, SP, #8 ; remove space for 2 variables

```
<table>
<thead>
<tr>
<th>Variable</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>LR</td>
<td>SP</td>
</tr>
<tr>
<td>birthYear</td>
<td>SP+4</td>
</tr>
<tr>
<td>birthMonth</td>
<td>SP+8</td>
</tr>
<tr>
<td>birthDay</td>
<td>SP+12</td>
</tr>
</tbody>
</table>
```
Example - return and discard parameters

<table>
<thead>
<tr>
<th>LR</th>
<th>← SP</th>
</tr>
</thead>
<tbody>
<tr>
<td>birthYear</td>
<td>SP+4</td>
</tr>
<tr>
<td>birthMonth</td>
<td>SP+8</td>
</tr>
<tr>
<td>birthDay</td>
<td>SP+12</td>
</tr>
</tbody>
</table>

LDR PC, [SP], #16 ; return and discard 3 parameters
Question: ARM code for method body

<table>
<thead>
<tr>
<th>ageInSomeYear</th>
<th>← SP</th>
</tr>
</thead>
<tbody>
<tr>
<td>someYear</td>
<td>SP+4</td>
</tr>
<tr>
<td>LR</td>
<td>SP+8</td>
</tr>
<tr>
<td>birthYear</td>
<td>SP+12</td>
</tr>
<tr>
<td>birthMonth</td>
<td>SP+16</td>
</tr>
<tr>
<td>birthDay</td>
<td>SP+20</td>
</tr>
</tbody>
</table>

ageInSomeYear = birthMonth + birthDay;
someYear = ageInSomeYear + birthYear;
Load/Store Multiple

ARM has “Store/Load Multiple” instructions to move several register contents to/from memory with a single instruction.

- May apply to any subset of 1-16 registers
- More efficient than using many instructions
- Limited in addressing options: only a single base register

e.g.:
STMFD SP!, {R0, R1, R3-R6}; push several registers
LDMFD SP!, {R0, R1, R3-R6}; pop several registers
(don’t worry about the “FD” part)

Two main uses:
- Saving/restoring registers to create working space
- Copying blocks of memory around

We are only interested in the first of these.
Saving Registers

The assembler has pseudo-instructions which save typing:

\begin{verbatim}
PUSH {R0, R1, R3-R6} ; push several registers
POP  {R0, R1, R3-R6} ; pop several registers
\end{verbatim}

Which builds a stack in the ‘traditional’ way.

- The listed register contents are moved (6 in this case)
- The memory used is consecutive; the lowest numbered register always corresponds to the lowest address. (i.e. the top of the stack)

If an expression is really complicated, we may need to save/restore registers to evaluate it.

Can save registers at start of method – “callee saved” (as in previous lecture) or before call – “caller saved”
We now have multiple uses for our stack

- method variables
- callee saved registers
- LR
- parameters
- caller saved registers
Question: redo the printAgeHistory example

The code used R0 and R1. Write ARM code to save R0 and R1 at the start of the method and restore them at the end.

Draw the stack-frame now that we are using extra memory locations for R0 and R1.
ARM Procedure Call Standard (APCS)

A **standard** on how to use registers in real programs

R0-R3: parameter/result passing (extra arguments are stacked)
anyone can use, but not saved across call (caller saved)

R4-R8, R10, R11: temporaries/locals (callee saved)

R9, R12: temporaries (caller saved)

SP, LR, PC: special purpose

(lots more details!)

We do not need to adhere to this standard in this unit!
Global Variables

In Java, variables that are “static”
– must exist throughout the run-time of the program
– can never have more than 1 copy

Simplest implementation: `DEFW` !
Summary of key points

Using Stack-Frames for:

– Method Parameters (Arguments)

– Method Variables

– Saving Registers

Static Variables

More ARM instructions
– Load & Store Multiple: LDM(FD) STM(FD)
Your Questions
Glossary

Local (method) variable
Global variable
Stack frame
STMFD instruction
LDMFD instruction
Callee method
Caller method
Caller-saved register
Callee-saved register
For next time

Explain the differences between LDR instructions & LDM instructions. (4 marks)

Explain the effect of the following ARM instruction on the registers involved and on memory: (2 marks)

STMFD SP!, {R0, R3–R5}

A method is passed two integer arguments via the stack, which it adds together, and returns the result in R0. The method also puts the Link Register on the stack (e.g. because it also calls “println” – but don’t include the code for this call in your answer). Give the ARM code required for a simple implementation of the method and draw a diagram of the resulting stack frame. (4 marks)
When implementing a method call on the ARM, its parameters, return address (link), and variables can be stored on the stack and accessed via the SP register.

Draw an example “stack frame” that includes each of these things.

Assuming the stack layout in your diagram, give example ARM code for:

– calling a method, including stacking parameters;
– the start of a method;
– using a parameter inside the method;
– using a method variable (stored on the stack);
– returning from the method.

(6 marks)
A method is given the address of a table of integers in R1 and the number of values in the table in R2 as parameters and returns the total of all the numbers in the table in R0. Give the ARM code for the method, assuming that the link register is saved on the stack. Your code must leave the argument registers and any other registers you may use, unchanged. Therefore, your code may need to save and restore registers appropriately.

(6 marks)