From last 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)
COMP15111: Introduction to Architecture

Lecture 11: Switch

Dr. Javier Navaridas

School of Computer Science, University of Manchester

Autumn 2016
Overview & Learning Outcomes

Java Switch
– Implemented as: if ... else if ...
– Using a Table

Signed & Unsigned integers
switch (number) {
  case 0:
    System.out.print("0");
    break;
  case 2:
    System.out.print("2");
    break;
  case 4:
    System.out.print("4");
    break;
  default:
    System.out.print("?");
    //break;
}
Question: ARM code for if-else version?

Note: `System.out.print("2");` becomes: `MOV R0, #'2'` then `SVC 0`
Observations

+ We already know how to do if...else...

– Speed: a large switch statement can have many cases so we may have to make many tests to get to cases near the bottom

Instead . . .

We want to use the value of the switch variable to branch directly to the correct piece of code.
Use a table (list) holding the addresses of the pieces of code

<table>
<thead>
<tr>
<th>Address</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>System.out.print(&quot;0&quot;);</td>
</tr>
<tr>
<td>1</td>
<td>System.out.print(&quot;2&quot;);</td>
</tr>
<tr>
<td>2</td>
<td>System.out.print(&quot;4&quot;);</td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>System.out.print(&quot;?&quot;);</td>
</tr>
</tbody>
</table>

The table has one word for each possible value of the variable.

Use the value of the variable to get a particular address from the table.
if (number<0 || number>4) B default;
PC= number’th word from table;
table:
    DEFW case0 ;  word 0 of table at table+0
    DEFW default ; 1 +4
    DEFW case2 ; 2 +8
    DEFW default ; 3 +12
    DEFW case4 ; 4 +16

case0: System.out.print("0");
    B end;
case2: System.out.print("2");
    B end;
case4: System.out.print("4");
    B end;
default: System.out.print("?");
end
ARM Code for Cases

case0  MOV  R0,#'0'
       SVC  0
       B    end

  case2  MOV  R0,#'2'
       SVC  0
       B    end

  case4  MOV  R0,#'4'
       SVC  0
       B    end

  default  MOV  R0,#'?'
       SVC  0

  end    . . .
ARM Code for start of Switch

LDR R0, number
CMP R0, #0
BLT default
CMP R0, #4
BGT default

ADR R1, table
LDR PC, [R1,R0,LSL #2] ; table + (number*4)

table DEFW case0 ; address = table + 0
DEFW default ; + 4
DEFW case2 ; + 8
DEFW default ; + 12
DEFW case4 ; + 16
Question

What is the ARM code for the start of this switch statement:

```java
switch (number) {
    case -1:
        System.out.print("-1");
        break;
    case 0:
        System.out.print("0");
        break;
    case 1:
        System.out.print("1");
        break;
    default:
        System.out.print("?");
        //break;
}
```
Observations

– Overhead of table lookup

+ Table size doesn’t affect speed

This is a simple example of using a table of addresses

We do something similar:
– to implement objects
– to handle peripherals
– to implement libraries (classes) of useful methods
  etc.
Every value is represented as a bit-pattern

e.g. bit-patterns for 0 and \texttt{false} may be the same

Java compiler checks types for us

ARM (assembler) doesn’t

e.g. can add characters (remember last example class?)
Signed & Unsigned

assume 8-bit numbers (instead of 32-bit) so 256 \( (2^8) \) different values

**Unsigned** values 0 to 255 (base-2 notation)

**Signed** values –128 to +127 (2’s-complement)

same representation for 0 to 127:
0x00 to 0x7F (0000 0000 to 0111 1111)

0x80 to 0xFF (1000 0000 to 1111 1111) represent:
– unsigned values 128 to 255
– signed values –128 to –1
Bit-patterns & numbers they can represent

8-bit-patterns:
0x00 0x01 ... 0x7E 0x7F 0x80 0x81 ... 0xFE 0xFF

signed values:
0 1 ... 126 127 -128 -127 ... -2 -1

unsigned values:
0 1 ... 126 127 128 129 ... 254 255

32-bit-patterns:
0x00000000 ... 0x7FFFFFFFFF 0x80000000 ... 0xFFFFFFFF

signed values:
0 ... $2^{31} - 1$ $-2^{31}$ ... -1

unsigned values:
0 ... $2^{31} - 1$ $2^{31}$ ... $2^{32} - 1$
Table Optimisation: initial tests

check (signed) value: $0 \leq n \leq 4$
i.e. 0 to 4 valid, every other value invalid

signed: fail if $n > 4$ (i.e. 0x00000005 to 0x7FFFFFFF)
or $n < 0$ (i.e. 0x80000000 to 0xFFFFFFFF)

unsigned: fail if $n > 4$ (i.e. 0x00000005 to 0xFFFFFFFF)

CMP R0, #0
BLT default signed test “Less Than”
CMP R0, #4
BGT default signed test “Greater Than”
→
CMP R0, #4
BHI default unsigned test “Higher than”
Improved Switch start sequence

LDR R0, number

CMP R0, #4
BHI default ; number < 0 OR number > 4

ADR R1, table
LDR PC, [R1,R0,LSL #2] ; table + (number*4)

table DEFW case0 ; address = table + 0
DEFW default ; + 4
DEFW case2 ; + 8
DEFW default ; + 12
DEFW case4 ; + 16
Question – is there a better answer?

(repeat previous question, but try adding 1 to number)

switch (number+1) {
    case 0:
        System.out.print(”–1”);
        break;
    case 1:
        System.out.print(”0”);
        break;
    case 2:
        System.out.print(”1”);
        break;
    default:
        System.out.print(”?”);
        //break;
}
Summary of key points

Java Switch
  – Implemented as: if ... else if ...
  – Using a Table

Signed & Unsigned integers
Your Questions
Glossary

Java switch
Java case
Java default
Java break
Table
Signed number
Unsigned number
BHI instruction
For next time

Explain how a table of code addresses ("jump table") can be used in the implementation of a switch statement. (4 marks)
Exam Questions

Translate the following Java statements, which are just part of a much larger program, into an equivalent sequence of ARM instructions. You should assume that the integer variables w,x,y,z are in memory and can be accessed just by using their name in a load or store instruction. Try to make your code as efficient as possible. (10 marks)

```java
while (x > 2) {
    y = (x - z) * y;
    switch (y) {
    case 0: w = w + 1; break;
    case 1: y = y - 1; break;
    default: x = 1 - x;
    }
}
```

// lots more code following