From last time

Explain how a table of code addresses ("jump table") can be used in the implementation of a switch statement. (4 marks)
COMP15111: Introduction to Architecture

Lecture 12: Booleans

Dr. Javier Navaridas

School of Computer Science, University of Manchester

Autumn 2016
Overview & Learning Outcomes

Java: booleans, operators etc.

Lazy operators

ARM code

more ARM instructions
Booleans

```java
int w, x, y, z;
if (w < x) { . . . } 

if (w < x & w > y) { . . . }; Logical AND
if (w < x | w > y) { . . . }; Logical OR

boolean b1, b2, b3, b4;
b1 = w < x;
b2 = w < x & w > y;
b3 = w < x | w > y;
b4 = (b1 | !b2) & b3 ^ b4; ! NOT, ^ XOR
if (b4) { . . . }

w = (w | ~x) & y ^ z; Bitwise vs Logical
```
Representing Boolean Variables

Tradeoff: Space vs. Time

Boolean (true/false) = 1 bit
→ could pack 8 booleans into a byte

Extracting 1 bit from a byte usually takes extra instructions

Normally don’t use lots of booleans, but do want fast code,
→ use a byte for each boolean

Usual representation: false is 0 (0x00), true is 1 (0x01)
(although you can’t see this directly in Java)

For interest:
In the C language there is no ‘boolean’ type and this is explicit!
Anything non-zero is interpreted as true.
Boolean Expressions

e.g.

```java
if (x == 0)
    System.out.println("x is zero");
```

Doesn’t need explicit boolean variable
– just CMP + BNE (as lecture 6)

```asm
LDR R0, x
CMP R0, #0
BNE false
...
```
Question

\[ b = (x == 0); \]
ARM “Bitwise” Logical Operations

e.g. \[ R1 = 0xFF00FF00 \]
\[ R2 = 0x0000FFFF \]

\[ \text{AND } R0, R1, R2; \quad R0 = 0x0000FF00 \]
\[ \text{ORR } R0, R1, R2; \quad R0 = 0xFF00FFFF \]
\[ \text{EOR } R0, R1, R2; \quad R0 = 0xFF0000FF; \quad \text{(Exclusive OR, XOR, !)=} \]
\[ \text{BIC } R0, R1, R2; \quad R0 = 0xFF000000; \quad \text{(Blt Clear) i.e. } R1 \text{ AND } R2 \]

\[
\begin{array}{cccccc}
A & B & \text{AND}=A \& B & \text{OR}=A | B & \text{EOR}=A ^{\wedge} B & \text{BIC}=A \& !B \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 1 & 0 \\
1 & 0 & 0 & 1 & 1 & 1 \\
1 & 1 & 1 & 0 & 0 & 0 \\
\end{array}
\]

\text{MVN (MoVe Not) is bitwise } \sim \text{ (not)}
Logical operations on booleans

We are using 0x00 and 0x01 for booleans, so:

& → AND  
| → ORR  
^ → EOR  
! → EOR with #1
Examples of logical operations on booleans

\[ b_4 = (b_1 \mid \neg b_2) \& b_3 \oplus b_4; \]

\[ ... \]

\[ \text{if} \ (b_4) \ ... \]

\[ \text{LDRB} \ R0, \ b1 \]

\[ \text{LDRB} \ R1, \ b2 \]

\[ \text{EOR} \ R1, \ R1, \ #1; \neg b_2 \]

\[ \text{ORR} \ R0, \ R0, \ R1; \mid \]

\[ \text{LDRB} \ R1, \ b3 \]

\[ \text{AND} \ R0, \ R0, \ R1; \& \]

\[ \text{LDRB} \ R1, \ b4 \]

\[ \text{EOR} \ R0, \ R0, \ R1; \lor \]

\[ \text{STRB} \ R0, \ b4 \]

\[ ... \]

\[ \text{LDRB} \ R0, \ b4 \]

\[ \text{CMP} \ R0, \ #0 \]

\[ \text{BEQ} \ false \]
b4 = (b1 & b2) | !b3;
Booleans from Comparisons

\[ b = (x < y \land y < z) \lor x > z; \quad (x, y, z \text{ ints, } b \text{ boolean}) \]

LDR R0, x
LDR R1, y
LDR R2, z
CMP R0, R1 ; x < y
MOVLT R3, #1 ; x < y true
MOVGE R3, #0 ; x < y false
CMP R1, R2 ; y < z
MOVLT R4, #1 ; y < z true
MOVGE R4, #0 ; y < z false
AND R3, R3, R4 ; &
CMP R0, R2 ; x > z
MOVGTE R4, #1 ; x > z true
MOVLT R4, #0 ; x > z false
ORR R3, R3, R4 ; |
STRB R3, b
Optimisation

Let’s look at part of the code a bit more closely:

```assembly
MOVLT R4, #1
MOVGE R4, #0
AND R3, R3, R4
```

This is the same as:

```assembly
ANDLT R3, R3, #1
ANDGE R3, R3, #0
```

For any b: b & true = b; b & false = false
so this is the same as:

```assembly
MOVGE R3, #0
```
Question

Optimise:

MOVGT R4, #1
MOVLE R4, #0
ORR R3, R3, R4
Optimised code

LDR   R0, x
LDR   R1, y
LDR   R2, z
CMP   R0, R1 ; x<y
MOVLT R3, #1
MOVGE R3, #0
CMP   R1, R2 ; y<z

MOVGE R3, #0 ;

CMP   R0, R2 ; x>z

MOVGT R3, #1 ;

STRB  R3, b

4 instructions shorter than the original.
Question

\[ b = (x > 7) \mid (x < 4); \]
Java lazy operators

&& and || are like & and | but “lazy”

– As an optimisation e.g.:

    var1 = exp1 && exp2;
    var2 = exp1 || exp2;

* If exp1 is false, var1 will always be false
* If exp1 is true, var2 will always be true
There is no point in evaluating exp2 in either of these cases

– As a necessity e.g.:

    if ((x == 0) || (y/x > 100)) ...
    if ((x == 0) | (y/x > 100)) ...

If x == 0, y/x will give an ArithmeticException
Example

\[ b = (x < y \land y < z) \lor x > z; \]

Rewrite to test \( x < y \) first:

```plaintext
if (x < y) b = (x < y \land y < z) \lor x > z;
else \quad b = (x < y \land y < z) \lor x > z;
```

```plaintext
if (x < y) b = (true \land y < z) \lor x > z; \quad x < y \text{ true}
else \quad b = (false \land y < z) \lor x > z; \quad x < y \text{ false}
```

```plaintext
if (x < y) b = (y < z) \lor x > z; \quad // true \& (\ldots) \text{ is (\ldots)}
else \quad b = (false) \lor x > z; \quad // false \& (\ldots) \text{ is false}
```

```plaintext
if (x < y) b = (y < z) \lor x > z; \quad // false \mid (\ldots) \text{ is (\ldots)}
else \quad b = x > z;
```
Example ctd.

\[
\text{if (x<y) } b = (y<z) \lor x>z; \\
\text{else } b = x>z;
\]

Rewrite to test y<z next:

\[
\text{if (x<y) }
\begin{cases}
\text{if (y<z) } b = \text{true } \lor x>z; \\
\text{else } b = \text{false } \lor x>z;
\end{cases}
\text{else } b = x>z;
\]

\[
\text{if (x<y) }
\begin{cases}
\text{if (y<z) } b = \text{true}; \quad \text{// true } \lor (...) \text{ is true} \\
\text{else} \quad b = x>z; \quad \text{// false } \lor (...) \text{ is (...)}
\end{cases}
\text{else } b = x>z;
\]

if both x<y and y<z then b=true, 
but if either is false then b=x>z
ARM code for Example

if both $x < y$ and $y < z$ then $b = \text{true}$, but if either is false then $b = x > z$

LDR R0, x
LDR R1, y
LDR R2, z
CMP R0, R1 ; $x < y$
BGE xz ; if $x < y$ false then $b = x > z$
CMP R1, R2 ; $y < z$
MOVLT R3, #1 ; if both $x < y$ and $y < z$ then $b = \text{true}$
BLT end ; if $y < z$ false then $b = x > z$
xz
CMP R0, R2 ; $x > z$
MOVLE R3, #0
MOVGT R3, #1
end
STRB R3, b
Question

\[ b = (x == 0) \mathbin{||} \text{oops}(x) ; \]

```java
boolean oops(int x) ...

1st parameter & result in R0
```
Summary of key points

Java: booleans, operators, lazy operators, etc.

ARM code: AND, ORR, EOR, BIC, MVN

Next time . . .
Glossary

Boolean
Java operators & | ! ^ ~ && ||
Result of a comparison/condition
Lazy operators
Eager operators
Logical operators
Bit-wise operators
Exclusive or (EOR, XOR)
AND instruction
ORR instruction
EOR instruction
BIC instruction
MVN instruction
Exam Questions

What are the results (in hexadecimal form) of the following bit-wise logical operations: (2 marks)
- \( 0x7e \ & \ 0x21 \)
- \( 0x7e \ ^{\wedge} \ 0x21 \)

If register R1 contains the value \( 0x5c \) and register R2 contains the value \( 0x6a \), what are the results (in hexadecimal form) of the following ARM instructions: (2 marks)
- \texttt{EOR R0, R1, R2} \\
- \texttt{AND R3, R1, R2}