From last time

Explain the similarities & differences between the ARM instructions 
\texttt{SUB R0, R1, R2} & \texttt{RSB R0, R1, R2} \ (2 \text{ marks})

Explain the similarities & differences between the ARM instructions 
\texttt{CMP R0, R1} & \texttt{CMN R0, R1} \ (2 \text{ marks})

Explain why it can be useful to have an “RSB” operation as well as an “SUB” operation in the ARM instruction set. Give an example to illustrate your answer. \ (2 \text{ marks})
COMP15111: Introduction to Architecture
Lecture 6: If and While

Dr. Javier Navaridas

School of Computer Science, University of Manchester

Autumn 2016
Overview & Learning Outcomes

Java → ARM code: control constructs
- if ( ... ) ...
- if ( ... ) ... else ...
- while ( ... ) ...
- for ( ... ) ...

Optimising loops:
- Making best use of registers
- More features of the ARM instruction set
Example if-statement from cash-till program

if total ≥ 20 pounds then
deduct 1 pound from total

Equivalent Java:

if (total >= 2000) {
    total = total - 100;
}

Explain to computer:

if total is not ≥ 2000
    skip over action

otherwise
    total = total - 100

(whatever comes next)
A generalised Java if-statement

if (condition) {
    action;
}

→ pattern of ARM instructions:

if condition is false
    skip over action
otherwise
    perform action
(whatever comes next)
Evaluating the condition

- evaluate arithmetic expressions
- compare
- (inverted) conditional branch
  e.g. if (a == b) → BNE skip
  and if (a < b) → BGE skip

  e.g. if (a + b >= c * d) { e = f; }
  LDR R0, a
  LDR R1, b
  ADD R0, R0, R1 ; R0 = a + b
  LDR R1, c
  LDR R2, d
  MUL R1, R1, R2 ; R1 = c * d
  CMP R0, R1
  BLT skip ; not >=
  LDR R0, f
  STR R0, e
  skip . . .
A generalised Java if-else-statement

```java
if (condition) {
    action;
} else {
    alternative;
}
```

- If condition is false, skip over action.
- Otherwise, perform action and skip over alternative.
- Perform alternative.
- (whatever comes next)
Why not the other way around?

```java
if (condition) {
    action;
} else {
    alternative;
}
```

- If condition is true, skip over alternative.
- Otherwise, perform alternative, skip over action, perform action, and then (whatever comes next).

Diagram:
- If condition is true: skip over alternative.
- Otherwise: perform alternative, skip over action, perform action, and then (whatever comes next).
Example if-else-statement

```java
if (a == b) {
    c = d;
} else {
    e = f;
}
```

```
LDR R0, a
LDR R1, b
CMP R0, R1
BNE skip ; not ==
LDR R0, d
STR R0, c
B both
skip
LDR R0, f
STR R0, e
both . . .
```
Question

Give ARM code for:

```c
if (a == b * c)  
a = b;
else  
b = b - 1;
```

Assume a, b, c are int variables.
Answer
Question

Give ARM code for:

```java
if (people > 300)
    watts = 500;
else if (people > 100)
    watts = 150;
else
    watts = normal;
```

Assume watts, people, normal are int variables.

Note: 300, 500, 100, 150 will all fit into ARM short literals.
Answer
A generalised Java while-statement

```java
while (condition) {
    action;
}
```

- if condition is false
- otherwise perform action
- (whatever comes next)
Example while-statement

```c
while (a < b) {
    a = a * 2;
}
```

```assembly
start  LDR  R0, a
    LDR  R1, b
    CMP  R0, R1
    BGE  skip ; not <
    MOV  R1, #2
    MUL  R0, R0, R1 ; a in R0
    STR  R0, a
    B    start
skip  . . .
```

(Not the most efficient compilation!)
Faster Loops

Body of while-statement obeyed repeatedly

→

Worth improving the code for the body
Move the test?

```assembly
B   start       ; Branch to (near) end
loop  MOV R1, #2  ; Start of loop 'body'
       MUL R0, R0, R1 ; a is in R0
       STR R0, a
start LDR R0, a  ; Continuation test
       LDR R1, b
       CMP R0, R1
       BLT loop     ; Close loop
...
```

Same number of lines of code
1 extra branch instruction obeyed to start
Instructions obeyed per repetition: 8 → 7
(2 branch instructions → 1)

We won’t do this – we will save the branch another way later!
Use registers instead of memory

- load a, b, and 2 into registers at start
- save a to memory at the end

```
start LDR R0, a
    LDR R1, b
    CMP R0, R1
    BGE skip
    MOV R1, #2
    MUL R0, R0, R1
    STR R0, a
    B start
skip . . .
```

```
start CMP R0, R1
    MOV R2, #2
    start CMP R0, R1
    BGE skip
    MUL R0, R0, R2
    B start
skip STR R0, a
```

Instructions obeyed per repetition: 8 → 4
(needs extra register to hold “2”)
Simplify Arithmetic

- change “a = a * 2” to “a = a + a”

```assembly
LDR R0, a
LDR R1, b
start CMP R0, R1
BGE skip
ADD R0, R0, R0
B start
skip STR R0, a
```

Don’t need R2 any more!
Question

Give ARM code for:

while (side * side * side < vol)
    side = side + 1;

Assume side, vol are int variables.
Conditional Instructions

Every ARM instruction can be made conditional.

Like branches: \( B \rightarrow \text{BEQ, BNE, BLT etc.} \)

Also have (e.g.) \( \text{MOVEQ, LDRNE, ADDLT, CMPGT etc.} \)

(Condition encoded in first 4 bits of every instruction)

This is an unusual feature of ARM. Most processors have only conditional branches – other instructions always unconditional – occasionally conditional ‘move’ as well
Previous example revisited

```
start CMP R0, R1 ; Compare a and b
   BGE skip ; if a≥b finish
   MUL R0, R0, R2 ; else multiply a by 2
   B start ; and repeat
skip ...  
becomes:

start CMP R0, R1 ; Compare a and b
   MULLT R0, R0, R2 ; if a<b multiply a by 2
   BLT start ; if a<b repeat
skip ... ; else finish
```

Instructions obeyed per repetition: $4 \rightarrow 3$
(again, saving a branch instruction by getting a conditional branch at the end of the loop)
(+ 1 extra MULLT ‘obeyed’ (ignored) at end)
Another example: Powers of 2

```plaintext
a = 1;
while (a * 2 > 0) {
    a = a * 2;
}

MOV R0, #1 ; keep a in R0
MOV R1, #2 ; keep 2 in R1

start MUL R2, R0, R1 ; while (a * 2 > 0)
CMP R2, #0 ; ...
BLE skip ; ...
MOV R0, R2 ; a = a * 2
B start ; repeat loop

skip STR R0, a ; store a
```

5 instructions per repetition
Powers of 2 (version 2)

Use conditional instructions:

```assembly
MOV R0, #1 ; keep a in R0
MOV R1, #2 ; keep 2 in R1
start MUL R2, R0, R1 ; while (a * 2 > 0)
CMP R2, #0 ; ...
MOVGT R0, R2 ; a = a * 2
BGT start ; repeat loop
skip STR R0, a ; store a
```

4 instructions per repetition
Question

Give the **best** ARM code that you can, using conditional arithmetic etc, for:

```plaintext
while (side * side * side < vol)
    side = side + 1;
```

Assume side, vol are int variables.
Comparing with zero: ...S

Arithmetic instructions (not LDR or STR) can also compare their result with 0

e.g.

```
ADD R2, R0, R1
CMP R2, #0
```

can become:

```
ADDS R2, R0, R1
```

Note: **MULS** only for *some* conditions . . . not >, < etc.

Note: Can use conditional instruction & “S” together (e.g. ADDNES)

On many (non-ARM) computers, arithmetic & load instructions always do this (details vary).
Optimised ‘for’ loop

In Java (or C) you may see something like:

```java
for (i = 1; i <= 10; i = i + 1) {
    action;
}
```

often this may be rearranged as:

```java
for (i = 10; i > 0; i = i - 1) {
    action;
}
```

i.e. count down instead of up

(Convince yourself that both of these iterate 10 times)
Optimised ‘for’ loop

for (i = 10; i > 0; i = i - 1) {
    action;
}

MOV R0, #10 ; 'i'
B start ;
loop ... ; action
... ;
start SUBS R0, R0, #1 ; count and test
BGT loop ; i > 0
... ;

This ‘trick’ usable on many processors/languages. Compilers will typically optimise for this when possible.

You may see other people’s loops counting down rather than up; this will usually be why.
Question

Give the best ARM code that you can, using conditional arithmetic etc., for: (x, y & z are positive int variables)

```c
x = 0;
while (y - z >= 0) {
    y = y - z;
    x = x + 1;
}
```

MOV R0, #0; R0 = x
LDR R1, y
LDR R2, z
start SUBS R3, R1, R2
MOVGE R1, R3
ADDGE R0, R0, #1
BGE start
STR R0, x
STR R1, y
Summary of key points

Java → ARM code: control constructs
– condition: extra register for sub-expressions
– if: if condition \texttt{false} skip over “then”
– if-else: ... skip over “then” to “else”; B end after “then”
– while $\approx$ if + B start

Optimising loops: small improvements obeyed repeatedly
– Move \textit{invariant} code (e.g. load, store) out of loop
– Simplify code inside loop

More features of the ARM instruction set
– Making any instruction conditional e.g. ADD\texttt{NE}
– Avoiding CMP \ldots , #0 e.g. ADD\texttt{S}
Your Questions
Glossary

Condition
Inverting a condition
Skip
Loop
Loop body
Conditional instruction
ADDNE instruction
ADDS instruction
ADDNES instruction
Overflow
GCD or HCF
For next time

What are the differences between these three ARM instructions: (3 marks)

ADD   R1, R2, R3
ADDS  R1, R2, R3
ADDEQ R1, R2, R3

Explain how using “S” (as for ADDS above) can sometimes help to make a code sequence shorter and faster, giving an example. (3 marks)

Explain how using conditional instructions can sometimes help to make a code sequence shorter and faster, giving an example. (3 marks)
Exam Questions

For both questions (1) & (2), translate the statements, which are just part of a much larger Java program, into equivalent ARM instructions. The integer variables w, x, y, z are in memory and can be accessed by name in load and store instructions. Make your code as efficient as possible. (10 marks each)

1) while (x - y > 0) {
    if (x < -y)
        {y= y - x; x= -x;}
    else
        {y= x * y; x= 2 - y;}
}

2) if (y < x)
   {y= y - x; x= y * x; z= -x;}
else
   {y= 3 - y; z= y + x; x= z * y;}

Self-study: Another loop optimisation example

Greatest Common Divisor / Highest Common Factor (GCD/HCF) of 2 +ve ints: largest +ve int that divides them without a remainder

e.g. GCD(30,12)=6, GCD(8,12)=4, GCD(9,10)=1

while (a != b) {
    if (a > b) {
        a = a - b;
    } else {
        b = b - a;
    }
}

Try the examples by hand e.g. a=30, b=12
GCD (1)

- Keep variables in registers

```assembly
LDR R0, a ; keep a in R0
LDR R1, b ; and b in R1
start CMP R0, R1 ; while (a != b)
    BEQ skip
    CMP R0, R1 ; if (a > b)
    BLE else
    SUB R0, R0, R1 ; then a = a - b
    B both
else SUB R1, R1, R0 ; else b = b - a
both B start ; and repeat loop
skip STR R0, a ; store a
STR R1, b ; and b
```

6 or 7 instructions per repetition
GCD (2)

(Ignore boundaries between while + if-else)

- reuse comparison: CMP R0, R1
- avoid branch to branch: B both → B start

LDR  R0, a ; keep a in R0
LDR  R1, b ; and b in R1
start CMP  R0, R1 ;
BEQ  skip ; loop test
BLE  else ; if test
SUB  R0, R0, R1
B    start
else SUB  R1, R1, R0
B    start
skip STR  R0, a
STR  R1, b

5 instructions per repetition
GCD (3)

- Use conditional arithmetic

(be careful with exact conditions: test at start of loop means \( a \neq b \) within if-else)

```
LDR    R0, a            ; keep a in R0
LDR    R1, b            ; and b in R1
start  CMP   R0, R1
    SUBGT R0, R0, R1   ; if \( a > b \) then \( a = a - b \)
    SUBLT R1, R1, R0   ; if \( a \) (was) < \( b \) then \( b = b - a \)
    BNE    start       ; if \( a \) (was) \( \neq \) \( b \) then repeat
STR    R0, a
STR    R1, b
```

4 instructions per repetition