

## CS1021 Tutorial #4 Solution

### Flow Control and Pseudo-Code Translation

## 1 Translating Pseudo-code into ARM Assembly Language

Translate each of the following pseudo-code programs into ARM Assembly Language, making use of the CMP instruction and the conditional branch instructions.

- (a) Assume  $x$  is a signed value stored in R0.

```
if (x > 1)
{
    x = x + 5;
}
```

```
1  CMP R0, #1
2  BLE endif
3  ADD R0, R0, #5
4  endif
```

- (b) Assume  $x$  is stored in R0.

```
if (x == 0)
{
    x = 1;
}
else
{
    x = x * 2;
}
```

```
1  CMP R0, #0
2  BNE else
3  MOV R0, #1
4  B    endif
5  else
6  MOV R1, #2
7  MUL R0, R1, R0 ; not worried about efficiency here!
8  endif
```

- (c) Assume  $x$  is a signed value stored in R0 and  $y$  is stored in R1.

```
while (x < 0)
{
    y = y * x;
    x = x + 1;
}
```

```
1 while
2     CMP R0, #0
3     BGE endwh
4     MUL R1, R0, R1
5     ADD R0, R0, #1
6     B   while
7 endwh
```

- (d) Assume  $x$  is an unsigned value stored in R0 and  $y$  is stored in R1.

```
while (x > 5)
{
    y = y + (2 * x);
    x = x - 5;
}
```

```
1 while
2     CMP R0, #5
3     BLE endwh
4     MOV R2, #2
5     MUL R2, R0, R2 ; not worried about efficiency here!
6     ADD R1, R1, R2
7     SUB R0, R0, #5
8     B   while
9 endwh
```

- (e) Assume  $i$  is an unsigned value stored in R0 and  $y$  is stored in R1.

```
for (i = 0; i < 10; i = i + 1)
{
    y = y + (i * i);
}
```

```
1     MOV R0, #0
2 fori
3     CMP R0, #10
4     BHS efori
5     MUL R2, R0, R0
6     ADD R1, R1, R2
7     ADD R0, R0, #1
8     B   fori
9 efori
```

- (f) Assume a, b and c are unsigned values stored in R4, R5 and R6 respectively.

```
while (a + b < 100)
{
    a = a + 1;
    b = b + c;
}
```

```
1 while
2     ADD R7, R4, R5
3     CMP R7, #100
4     BHS endwh
5     ADD R4, R4, #1
6     ADD R5, R5, R6
7     B    while
8 endwh
```

- (g) Assume s is an unsigned value stored in R3, t is an unsigned value stored in R4 and r is an unsigned value stored in R5.

```
t = 0;
while (t < 5)
{
    s = 0;
    while (s < 10)
    {
        r = (t * 10) + s;
        s = s + 1;
    }
    t = t + 1;
}
```

```
1     MOV R6, #10
2     MOV R4, #0
3 whilet
4     CMP R4, #5
5     BHS ewhilet
6     MOV R3, #0
7 whiles
8     CMP R3, #10
9     BHS ewhiles
10    MUL R5, R4, R6
11    ADD R5, R5, R3
12    ADD R3, R3, #1
13    B    whiles
14 ewhiles
15    ADD R4, R4, #1
16    B    whilet
17 ewhilet
```

(h) Assume `ch` is an ASCII character code stored in `R1` and `v` is stored in `R0`.

```
if (ch >= '0' && ch <= '9')
{
    v = ch - '0';
}
else if (ch >= 'A' && ch <= 'F')
{
    v = ch - 'A' + 0xA;
}
else
{
    v = 0xFFFFFFFF;
}
```

```
1  CMP R1, #'0'
2  BLO elif
3  CMP R1, #'9'
4  BHI elif
5  SUB R0, R1, #'0'
6  B    endif
7  elif
8  CMP R1, #'A'
9  BLO else
10 CMP R1, #'F'
11 BHI else
12 SUB R0, R1, #'A'-0xA    ; OK as assembler will calculate constant
13 B    endif
14 else
15 MOV R0, #0xFFFFFFFF
16 endif
```

(What does this pseudo-code do?)

Hexadecimal character to value