Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

School of Computer Science and Statistics
CS1021 / Introduction to Computing I

CS1021 Tutorial #4 Solution
Pseudo-code and Flow Control

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 shown on the **ARM Conditional Branch Instructions** reference card.

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

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

```
        CMP     R0, #1
        BLE     endif
        ADD     R0, R0, #5
endif
```

(b) Assume x is stored in R0.

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

```
        CMP     R0, #0
        BNE     else
        MOV     R0, #1
        B       endif
else
        MOV     R1, #2
        MUL     R0, R1, R0      ; not worried about efficiency here!
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;
}
```

```
while
        CMP     R0, #0
        BGE     endwh
        MUL     R1, R0, R1
        ADD     R0, R0, #1
        B       while
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;
}
```

```
while
        CMP     R0, #5
        BLE     endwh
        MOV     R2, #2
        MUL     R2, R0, R2      ; not worried about efficiency here!
        ADD     R1, R1, R2
        SUB     R0, R0, #5
        B       while
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);
}
```

```
        MOV     R0, #0
fori
        CMP     R0, #10
        BHS     efori
        MUL     R2, R0, R0
        ADD     R1, R1, R2
        ADD     R0, R0, #1
        B       fori
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;
}
```

```
while
        ADD     R7, R4, R5
        CMP     R7, #100
        BHS     endwh
        ADD     R4, R4, #1
        ADD     R5, R5, R6
        B       while
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;
}
```

```
        MOV     R6, #10
        MOV     R4, #0
whilet
        CMP     R4, #5
        BHS     ewhilet
        MOV     R3, #0
whiles
        CMP     R3, #10
        BHS     ewhiles
        MUL     R5, R4, R6
        ADD     R5, R5, R3
        ADD     R3, R3, #1
        B       whiles
ewhiles
        ADD     R4, R4, #1
        B       whilet
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;
}
```

```
        CMP     R1, #'0'
        BLO     elsif
        CMP     R1, #'9'
        BHI     elsif
        SUB     R0, R1, #'0'
        B       endif
elsif
        CMP     R1, #'A'
        BLO     else
        CMP     R1, #'F'
        BHI     else
        SUB     R0, R1, #'A'-0xA        ; OK as assembler will calculate constant
        B       endif
else
        MOV     R0, #0xFFFFFFFF
endif
```

(What does this pseudo-code do?)

Hexadecimal character to value