

CSSE 132 – Introduction to Systems Programming
 Rose-Hulman Institute of Technology

ARM vs. C — Some examples

1 Simplified Examples

These examples assume all the variable data are stored in registers. This is not usually the case, but this helps illustrate basic C structures.

1.1 main function

```
1 int main() {
2     int x = 13;
3     int y = 14;
4     return x + y;
5 }
```

```
1 main:
2     mov w2, 13
3     mov w3, 14
4     add w0, w2, w3
5     ret
```

1.2 if statement

```
1 x = 10;
2
3 if ( x > 0 ) {
4
5     x = x - 1;
6 }
```

```
1     mov w3, 10      @ w3 holds 'x'
2     cmp w3, 0
3     ble .ENDIF     @ x>0 = !(x<=0)
4     sub w3, w3, 1
5 .ENDIF:
```

1.3 if/else statement

```
1 x = 10;
2
3 if ( x > 0 ) {
4
5     x = x - 1;
6
7 } else {
8     x = x + 1;
9 }
```

```
1     mov w3, 10      @ w3 holds 'x'
2     cmp w3, 0
3     ble .ELSE     @ x>0 = !(x<=0)
4     sub w3, w3, 1
5     b .ENDIF
6 .ELSE:
7     add w3, w3, 1
8 .ENDIF:
```

1.4 do/while loop

```

1 x = 10;
2 do {
3     x = x - 1;
4 }
5 while ( x > 0 );

```

```

1     mov w3, 10    @ w3 holds 'x'
2 .LOOP:
3     sub w3, w3, 1
4     cmp w3, 0
5     bgt .LOOP
6 .ENDLOOP:

```

1.5 while loop

```

1 x = 10;
2
3 while ( x > 0 ) {
4
5     x = x - 1;
6 }

```

```

1     mov w3, 10    @ w3 holds 'x'
2 .LOOP:
3     cmp w3, 0
4     ble .ENDLOOP
5     sub w3, w3, 1
6     b .LOOP
7 .ENDLOOP:

```

There's another way to compile the same while loop. This way looks more like the do-while translation to assembly:

```

1     mov w3, 10    @ w3 holds 'x'
2     b .TEST
3 .LOOP:
4     sub w3, w3, 1
5 .TEST:
6     cmp w3, 0
7     bgt .LOOP
8 .ENDLOOP:

```

1.6 for loop

There's lots of extra space in this for loop. Usually you'd write the for statement like this:

for (x = 10; x > 0; x--) { ... } but to better see how the bits of C map to ARM, spaces are added:

```

1 y = 0;
2 for ( x = 10;
3
4     x > 0 ;
5
6     x--      ) {
7     y = y + x;
8 }

```

```

1     mov w2, 0    @ w2 holds 'y'
2     mov w3, 10   @ w3 holds 'x'
3 .LOOP:
4     cmp w3, 0
5     ble .ENDLOOP @ stop if !(x>0)
6     add w2, w2, w3 @ y=y+x
7     sub w3, w3, 1 @ x--
8     b .LOOP
9 .ENDLOOP:

```

2 Load/Store Examples

These examples now assume variables are assigned a memory location. This means they are loaded and stored as necessary.

2.1 set elements in array

Assume `arr` is an array and its address is stored in `x4`:

```

1 arr[0] = 40;
2
3
4 arr[1] = 30;
5
6
7 arr[2] = arr[0] + arr[1];

```

```

1  mov w0, 40
2  str w0, [x4] @ x4 is addr of arr
3
4  mov w0, 30
5  str w0, [x4, 4]
6
7  ldr w0, [x4] @ arr[0]
8  ldr w1, [x4, 4] @ arr[1]
9  add w0, w0, w1 @ arr[0]+arr[1]
10 str w0, [x4, 8]

```

2.2 Local variables stored on the stack

When you declare variables in a C function, it makes space on the stack by moving `sp`, then assigns the variables locations there (much like entries in an array). Notice that `x = y`; requires both a load from memory and a store into memory since both variables are stored in memory!

In this example `x`, `y` and `tmp` are local variables stored on the stack:

```

1 void swap()
2 {
3
4  int x = 1;
5
6
7  int y = 16;
8
9
10 int tmp = x;
11
12
13 x = y;
14
15
16 y = tmp;
17
18
19 }

```

```

1 swap:
2  sub sp, sp, 12 @ make space
3
4  mov w3, 1
5  str w3, [sp, 0] @ x is sp+0
6
7  mov w3, 16
8  str w3, [sp, 4] @ y is sp+4
9
10 ldr w3, [sp, 0]
11 str w3, [sp, 8] @ tmp is sp+8
12
13 ldr w3, [sp, 4] @ put y into x
14 str w3, [sp, 0]
15
16 ldr w3, [sp, 8]
17 str w3, [sp, 4]
18
19 add sp, sp, 12 @ shrink stack
20 ret

```