Tutorial 9: Instruction Encoding and Decoding

Problem 1: Pseudo Instruction

Consider the ARM assembly code in Figure 1 Answer the following questions:
What is the real ARM instruction corresponding to the pseudo instruction at label L2?
What is the real ARM instruction corresponding to the pseudo instruction at label L3?
What is the real ARM instruction corresponding to the pseudo instruction at label L4?
What is the content of the register r0 in instruction (str r0, [r2, #0])?

| L1: | ldr r0, data+4 |
| L2: | ldr r1, data+8 |
|     | ldr r0, [r0, #0] |
|     | ldr r1, [r1, #0] |
|     | add r0, r0, r1 |
| L3: | adr r2, data |
| L4: | mov r3, #0x00ffffff |
|     | add r0, r0, r3 |
|     | str r0, [r2, #0] |
|     | ldmfd sp!, {pc} |

Figure 1: Sample ARM Assembly Code
Problem 2: Instruction decoding

Convert the ARM machine code in Figure 2 to a corresponding C Program

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0:</td>
<td>e5902000</td>
</tr>
<tr>
<td>4:</td>
<td>e3a00000</td>
</tr>
<tr>
<td>8:</td>
<td>e1a03000</td>
</tr>
<tr>
<td>c:</td>
<td>e1530002</td>
</tr>
<tr>
<td>10:</td>
<td>a1a0f00e</td>
</tr>
<tr>
<td>14:</td>
<td>e0800003</td>
</tr>
<tr>
<td>18:</td>
<td>e2833001</td>
</tr>
<tr>
<td>1c:</td>
<td>e1530002</td>
</tr>
<tr>
<td>20:</td>
<td>bafffffb</td>
</tr>
<tr>
<td>24:</td>
<td>e1a0f00e</td>
</tr>
</tbody>
</table>

Figure 2: The Sample ARM Machine Code
### Load Store Category:

<table>
<thead>
<tr>
<th>Cond</th>
<th>opcode</th>
<th>S</th>
<th>Rn</th>
<th>Rd</th>
<th>Immediate Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0: e5902000 1110 010 1100100000010000000000000</td>
<td>14=Always load/store immediate Pre index up (+ve offset) word transfer</td>
<td>no write back load Base reg=0 Destination reg=2 offset = 0</td>
<td>0 1 0000 0010 000000000000</td>
<td></td>
</tr>
</tbody>
</table>

### Decoded Instruction:

0: ldr r2, [r0, #0]

### Figure 5: The Decoding Process for the Load/Store Instruction

Figure 6 demonstrates the decoding process for the instructions in the data processing category. We divide these instructions into immediate and register sub-categories.

**Data Processing Category Immediate:**

<table>
<thead>
<tr>
<th>Cond</th>
<th>opcode</th>
<th>S</th>
<th>Rn</th>
<th>Rd</th>
<th>Immediate Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>4:</td>
<td>e3a00000 1110 001 1101 0 0000 0000 000000000000</td>
<td>14=Al 001 13=mov 0=(no S) 0=r0 0=r0 0=0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18:</td>
<td>e2833001 1110 001 0100 0 0011 0011 000000000001</td>
<td>14=AL 001 4=add 0=(no S) 3=r3 3=r3 1=1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Decoded Instructions:

4:   mov   r0, #0

18:   add   r3, r3, #1

**Data Processing Category Register:**

<table>
<thead>
<tr>
<th>Cond</th>
<th>opcode</th>
<th>S</th>
<th>Rn</th>
<th>Rd</th>
<th>Shift fields</th>
<th>Rm</th>
</tr>
</thead>
<tbody>
<tr>
<td>8:</td>
<td>e1a03000 1110 000 1101 0 0000 0011 00000000 0000</td>
<td>14=Al 000 13=mov 0=(no S) 0=r0 3=r3 0=(no shift) 0=r0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c:</td>
<td>e1a530003 1110 000 1010 1 0011 0000 00000000 0010</td>
<td>14=AL 000 10=cmp 1=(S) 3=r3 0=r0 0=(no shift) 2=r2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10:</td>
<td>a1a0f00e 1010 000 1101 0 0000 1111 00000000 1110</td>
<td>10=GE 000 13=mov 0=(no S) 0=r0 15=r15 0=(no shift) 14=r14</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14:</td>
<td>e0800003 1110 000 0100 0 0000 0000 00000000 0011</td>
<td>14=AL 000 4=add 0=(no S) 0=r0 0=r0 0=(no shift) 3=r3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1c:</td>
<td>e1a530003 1110 000 1010 1 0011 0000 00000000 0010</td>
<td>14=AL 000 10=cmp 1=(S) 3=r3 0=r0 0=(no shift) 2=r2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24:</td>
<td>e1a0f00e 1110 000 1101 0 0000 1111 00000000 1110</td>
<td>14=AL 000 13=mov 0=(no S) 0=r0 15=r15 0=(no shift) 14=r14</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Decoded Instructions:

8:   mov   r3, r0

c:   cmp   r3, r2

10:   movge r15, r14

14:   add   r0, r0, r3

1c:   cmp   r3, r2

24:   mov   r15, r14

### Figure 7: The Decoding Process for the Data Processing Instructions

Figure 7 demonstrates the decoding process for the single instruction in the control category.
Control Category: Cond       Branch  Signed offset in word
20:   bafffffb 1011  101  0      111111111111111111111011

Decoding Process:
20:   bafffffb 11=LT 101  0=Branchonly   -5= go back 5 words (20 bytes) from
current PC, or 3 words (12 bytes)

Decoded Instructions:
ble 0x14

Figure 7: The Decoding Process for the Data Processing Instructions

Putting back everything together, we will have the ARM assembly code in Figure 8
0:   e5902000        ldr     r2, [r0]
4:   e3a00000        mov     r0, #0
8:   e1a03000        mov     r3, r0
c:   e1530002        cmp     r3, r2
10:   a1a0f00e        movge   r15, r14
14:   e0800003        add     r0, r0, r3
18:   e2833001        add     r3, r3, #1
1c:   e1530002        cmp     r3, r2
20:   bafffffb        blt     0x14
24:   e1a0f00e        mov     r15, r14

Figure 8: The Decoded ARM Assembly Code

The ARM assembly code in Figure 8 can be re-written as symbolic ARM assembly code as
shown in Figure 9.
sum_arr:
   ldr     a3, [a1]
   mov     a1, #0
   mov     a4, a1
   cmp     a4, a3
   Loop:   movge   pc, lr
   add     a1, a1, a4
   add     a4, a4, #1
   cmp     a4, a3
   blt     Loop
   mov     pc, lr

Figure 9: The Decoded ARM Symbolic Assembly Code

The symbolic ARM assembly code in Figure 9 can be re-written as a C code as shown in Figure
10.
int sum_arr(int *num)
{
   int n, s=0;
   for (n=0; n<(*num); n++)
   {
      s = s + n;
   }
   return s;
}

Figure 10: The Decoded C Code

The C function in Figure 10 corresponds to the algorithm in (1).
**Problem 3: Literal Pool**

Consider the C function in Figure 11. Produce the symbolic and true ARM assembly and machine versions of this code.

```c
int sum(int s)
{
    int n = 0x00ff00ff,
    int m = 0xee00ee00;
    return (s + m + n);
}
```

**Figure 11: C-code for the Demonstration of the Literal Pool**

The code in Figure 12 is the symbolic ARM assembly version of the C code in Figure 11.

```assembly
ldr a2, =0x00ff00ff ; pseudo instruction
ldr a3, =0xee00ee00 ; pseudo instruction
add a2, a2, a3
add a1, a1, a2
```

**Figure 12: Symbolic Assembly Code for the Demonstration of the Literal Pool**

The code in Figure 13 is the real ARM assembly version of the symbolic code in Figure 12.

```assembly
ldr r1, [pc, #8]; real instruction
ldr r2, [pc, #8]; real instruction
add r1, r1, r2
add r0, r0, r1
```

**Figure 13: Real Assembly Code for the Demonstration of the Literal Pool**

The code in Figure 14 is the real machine version of the real ARM assembly code in Figure 13.

```assembly
0:   e59f1008    ldr r1, [pc, #8]; real instruction
4:   e59f2008    ldr r2, [pc, #8]; real instruction
8:   e0811002    add r0, r0, r2
c:   e0800001    add r0, r0, r1
```

**Figure 14: Machine Code for the Demonstration of the Literal Pool**