# Model 3: Translating Data-processing Instructions

Below are two examples of translating an instruction to machine code. The translation involves four steps i, ii, iii, iv. The **questions that come after** the examples will guide you through the examples.

## Example 1.

add R5, R0, R1 the instruction we are translating to machine code

add Rd, Rn, Rm for reference, the syntax for add (register operand)

Data-processing instruction format, where second operand is a register

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **cond** | **op** | **I / cmd / S** | **Rn** | **Rd** | **shamt5** | **sh** | **0** | **Rm** |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **cond** | **op** | **I / cmd / S** | **Rn** | **Rd** | **shamt5** | **sh** | **0** | **Rm** |
| 11102 | 002 | 0010002 | 0 | 5 | 0 | 0 | 0 | 1 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31:28 | 27:26 | 25:20 | 19:16 | 15:12 | 11:7 | 6:5 | 4 | 3:0 |
| **cond** | **op** | **I / cmd / S** | **Rn** | **Rd** | **shamt5** | **sh** | **0** | **Rm** |
| 1110 | 00 | 001000 | 0000 | 0101 | 00000 | 00 | 0 | 0001 |

**1110 00 001000 0000 0101 00000 00 0 0001**

**0xE0805001**

## Example 2.

sublt R2, R12, #400 the instruction we are translating to machine code

sublt Rd, Rn, imm for reference, the syntax for add (immediate operand)

Data-processing instruction format, where second operand is an immediate (specifically a right-rotated 8-bit immediate)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **cond** | **op** | **I / cmd / S** | **Rn** | **Rd** | **rot** | **imm8** |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **cond** | **op** | **I / cmd / S** | **Rn** | **Rd** | **rot** | **imm8** |
| 10112 | 002 | 1001002 | 12 | 2 | 14 | 0x19 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 31:28 | 27:26 | 25:20 | 19:16 | 15:12 | 11:8 | 7:0 |
| **cond** | **op** | **I / cmd / S** | **Rn** | **Rd** | **rot** | **imm8** |
| 1011 | 00 | 100100 | 1100 | 0010 | 1110 | 0001 1001 |

**1011 00 100100 1100 0010 1110 0001 1001**

**0xBE24C2E19**

1. How many bits are there in each field of a Data-processing instruction with a register operand?

cond: 4 op:2 I / cmd / S:1/4/1 = 6 Rn:4 Rd:4 shamt5:5 sh:2 0:1 Rm:4

1. How many bits are there in each field of a Data-processing instruction with an immediate operand?

cond:4 op:2 I / cmd / S:1/4/1 = 6 Rn:4 Rd:4 rot:4 imm8:8

1. What is the value for the **op** field in Data-processing instructions?

Is always 00 for data-processing instructions.

1. How is the value of the **cond** field determined for each of the two instructions?

1110 – ignored

1011 – signed less than

1. How is the value of **I, cmd, and S** determined for each of the two instructions?

I: if source (Src2) is immediate = 1, if register = 0

Cmd: Data processing instruction (add/sub)

S: Determines if

1. How were the registers translated to numbers?

Their register number is translated into binary. (R5 = decimal 5 = binary 5 = 0101)

1. For example 2, the constant 400 requires more than 8 bits. So then how do we encode the constant 400 in the machine code?

Uses rotation. Need to encode 400 as a rotation by an even number of digits.

400 -> 32 bit binary 00000…0000110010000.

How many left rotations to get 11001 into the lowest 8 bits? With 4 right rotations = 32 – 4 left rotations = 28 left rotations.

Now we have 00000…00011001.

You don’t lose 1’s and 0’s with rotations, they just hop to the other side.

28/2 = 14 left rotations by 2 digits (this is just how it was coded).

Imm8 = 00011001 = 0x19 = or as decimal

Rot = 14

**Pages 332 (condition) & 660 (operation)**

1. How are steps (ii) and (iii) related?

Step ii still has some decimal numbers, so step iii finishes the conversion to binary.

1. How are steps (iii) and (iv) related?

Step iv just gets rid of what each number means and translates to hexadecimal. It is what the computer actually receives and reads.

1. How are the two numbers in step (iv) related?

It is the binary and hexadecimal versions of the same number.

1. Give a short description (1 sentence or less) for each step.
   * 1. Step 1 sets up categories/format of what the instruction is doing.

* + 1. Step 2 fills in the categories of step 1 with numbers

* + 1. Step 3 converts all numbers to binary

* + 1. Step 4 removes the categories and puts all of the binary numbers in order. Also translates to hexadecimal.

# Read this!

Having an arbitrary mapping of instructions to binary numbers would be quite complex. What the computer architects did instead was quite intentional: they broke the instructions up into meaningful ***fields***, each specifying an attribute like an operand or the operation.

# Exercises

1. Translate the following instructions to machine code using the same steps as above.
   1. orreq R12, R1, R8

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **cond** | **op** | **I / cmd / S** | **Rn** | **Rd** | **shamt5** | **sh** | **0** | **Rm** |
| 0000 | 00 | 011000 | 1 | 12 | 0 | 0 | 0 | 8 |
| 0000 | 00 | 0 1100 0 | 0001 | 1100 | 00000 | 00 | 0 | 1000 |

as binary: 0000 00 011000 0001 1100 00000 00 0 1000

as hexadecimal: 0x181C008

* 1. eor R7, R3, #25600

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **cond** | **op** | **I / cmd / S** | **Rn** | **Rd** | **rot** | **imm8** |
| 1110 | 00 | 1 0001 0 | 3 | 7 | 11 | 0x1916 |
| 1110 | 00 | 100010 | 0011 | 0111 | 1011 | 00011001 |

000…0110010000000000

Shift right 10 = shift left 32 – 10 = 22 / 2 = 11

as binary: 1110 00 100010 0011 0111 1011 00011001

as hexadecimal: 0xE2237B19

# Extension Questions

1. The ARM instruction formats do not make full use of the 32 bits.
2. Give an example of a 32-bit code that is unused: that is, it does not correspond to any ARM instruction. Explain your answer.
3. Give an example of two 32-bit codes that are redundant: that is, they represent the *same* ARM instruction. By "same ARM instruction" we mean they both correspond to the same syntax. We **do not** mean two instructions with different syntax that achieve the same effect, such as

add R0, R1, #0

sub R0, R1, #0

1. Do you think having unused and redundant codes is a reasonable design choice for an instruction set architecture? Why? What is an alternative approach that would reduce waste? Can you think of disadvantages to that alternative?