### ARM Cheatsheet

### Layout

AREA <name>, CODE, READONLY

ENTRY

start

<code>

<more code>

...

SWI 0x11

END

Note: SWI 0x11 is not needed when trying stuff out in the Rolecks emulator.

### Commands

**<COMMAND> <arg1>, <arg2>**<tab>; <comment>

Note the space before the command.

The tab, semicolon and comment are optional.

Example:

MOV r0, #11 ; assign 11 to r0

**MOV**

Assign the value in the 2nd argument to the register in the 1st argument.

Examples

MOV r0, #11

Assign the number 11 to register 0.

MOV r0, r1

Assign the value in register 1 to register 0.

**ADD**

Example

ADD r5, r1, r2

Add the number stored in register 2 to the number stored in register 1. Store the result in register 5.

**SUB**

Example

SUB r2, r0, #1

Subtract the number 1 from the number stored in register 0. Store the result in register 2.

**ADR <register>, <label>**

Sets up a register to store a memory address. See the example on the next page.

**LDR**

Loads the value stored in a main memory location (RAM) into a register.

Examples

LDR r0, [r1]

Uses the value stored in register 1 to find a main memory location. It then loads the value stored at that main memory location into register 0.

Before r1 can be used to point to an address, it must be set up using ADR.

**STR**

The opposite of LDR. Copies a value held in a register into main memory.

Example

STR r0, [r1]

Stores the value held by register 0 in the main memory location pointed to by r1.

Before r1 can be used to point to an address, it must be set up using ADR.

**B <label>**

Makes a jump to another part of the program.

Example

B start

Makes the program jump to the 'start' label.

B function

Makes the program jump to the 'function' label.

**CMP**

Compares two values and then sets the CPSR depending on this comparison.

The CPSR is a special set of registers containing an N bit, an Z bit, a C bit and a V bit.

Example

CMP r1, r2

The Z and C bits are useful when branching in 'if statements' and 'for loops'. See below.

See **BNE** and **BHS**.

**BNE <label>**

Jumps to another point in the program, but only if the Z bit is set to 0. If you have compared two numbers using CMP, and the two numbers are not equal, then the jump is made. Useful for loops (see below). Opposite: BEQ

**BHS <label>**

Jumps to another point in the program, but only if the C bit is set to 1. If you have compared two numbers using CMP, and the first number is greater than or equal to the second number, then the jump is made. Opposite: BLO

**MVN**

The same as MOV, but all the bits in the value assigned are inverted.

Example:

MVN r0, r1

Assigns the value in register 1 to register 0, inverting all the bits.

Since numbers are represented in two’s complement, if you want to get the negative of a number, using MVN will get you the negative number one less than the one you expect. You will need to add 1 to the number to truly get the negative of the number.

Example

MVN r0, #5

ADD r0, r0, #1

r0 now equals ‘–5’ in two’s complement.

### For Loop

MOV r0, #0 Set up a counter.

loop Set up a label to jump back to.

*<more code>* Do stuff.

ADD r0, r0, #1 Add 1 to whatever is stored in r0 (store the result in r0 as well)

CMP r0, #10 Compare the counter (r0) to a number. This number is the total number of times the loop will run. In this case, the loop will run 10 times.

BNE loop Check the result of the comparison. If the counter and the total are not equal we are not finished looping. The BNE makes the program jump back to the ‘loop’ label.

*<more code>* If the two numbers were equal, we are done looping. The program continues with the next step without jumping.

### If Statement

CMP r0, r1 Compare r0 to r1.

BHS extra If r0 is greater than or equal to r1, the BMI makes the program jump to the ‘extra’ label.

B continue If r0 is less than r1, then the jump doesn’t happen. Instead, the B command jumps the program directly to ‘continue’, missing out the code under ‘extra’.

extra This is the ‘extra‘ label.

*<do stuff>* Extra stuff is done…

B continue B makes the program jump to the ‘continue’ label.

continue This is the ‘continue’ label. The program continues as normal.

*<more code>*

### Load & Store

This template loads a value from the NUMA memory location, and then stores that value in NUMB.

Please note: the rolecks emulator doesn’t recognise the DATA block, so this example won’t work in rolecks.

AREA loadstore, CODE, READONLY

ENTRY

start

ADR r1, NUMA Set r1 to store the memory address pointed to by NUMA

LDR r2, [r1] Load into r2 the value stored at the memory address pointed to by r1

*<do stuff>* Extra stuff is done…

ADR r3, NUMB Set r3 to store the memory address pointed to by NUMB

STR r2, [r3] Store the value in r2 into the memory address pointed to by r3

END

DATA data, DATA, READWRITE

NUMA #8 Sets up a memory location called NUMA, with a value of 8 stored in it

NUMB #0 Sets up a memory location called NUMB, with a value of 0 stored in it