# ARM 32-Bit Instructions

## 1 ARM Programmer's Model

| | |
|---|---|
| R0 | Scratch Register (by convention) |
| R1 | Scratch Register (by convention) |
| R2 | Scratch Register (by convention) |
| R3 | Scratch Register (by convention) |
| R4 | |
| R5 | |
| R6 | |
| R7 | Frame Pointer (by convention) |
| R8 | |
| R9 | |
| R10 | |
| R11 | |
| R12 | |
| R13 | Stack Pointer (SP) |
| R14 | Link Register (LR) |
| R15 | Program counter (PC) |

## 2 ARM Instructions

### 2.1 Data Movement Instructions

#### 2.1.1 LDR

(Load Register)
Loads a value into a register

**Example Usage** Loading a literal (hardcoded constant) value into a register

```
int main () {
  int i = 5; // allocate R0 to i
}
```

```
    .text
    .global main
main:
    ldr r0,=5
    bx lr
```

**Example Usage** Loading the value of a global variable into a register. The global variable lives in memory at some compiler-assigned address. We can refer to the address of the global variable by using the global variable's name. The first LDR instruction in the program below gets the compiler-assigned address of the global variable into R1. The second LDR goes out to memory at the address in R1 and reads four bytes into R0, copying the value of the global variable into R0.

```
int global_var = 77;
int main () {
  int i = global_var; // i lives in R0
}
```

```
    .text
    .global main
main:
    ldr r1,=global_var ; Get addr of global_variable in
    ldr r0,[r0]        ; Load data stored @ addr of globa
    bx lr

    .data
global_var:
    .word 77 ; // Initialize global_variable to 77
```

### 2.1.2 PUSH and POP

Save one or more register values on the stack. In general, any register that you use in a function should get pushed onto the stack at the beginning of the function and popped back off of the stack at the end. The only exception to this rule is R0, which is used to hold the function's return value. R0 should not get saved and restored.

PUSH and POP should also be used to save and restore the link register, which gets modified by function calls (BL instruction).

**Example Usage**

```
int main () {
  int i = 5; // allocate R0 to i
}
```

```
    .text
    .global main
main:
    push {r0} ; Save value in R0 so LDR doesn't overwrit
    ldr r0,=5 ; Set R0 to 5
    pop {r0}  ; Restore R0's old value from stack
    bx lr
```

### 2.1.3 MOV

(Move)

Copies a value from one register to another

**Example Usage**

```
int main () {
  int i = 5;   // i lives in R0
  int num = i; // num lives in R1
}
```

```
    .text
    .global main
main:
    ldr r0,=5 ; i <- 5
    mov r1,r0 ; num <- i
    bx lr
```