

Constants and literal pools



R. Ferrero, P. Bernardi

Politecnico di Torino

Dipartimento di Automatica e Informatica (DAUIN)

Torino - Italy

This work is licensed under the Creative Commons (CC BY-SA) License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/3.0/>



MOV

- It assigns a value to a register.
- The value can be:
 - the content of a register
 - a constant value.
- MOV cannot assign an instruction or data address

```
myCode    MOV r0, myData      ; error
```

```
          MOV r1, myCode      ; error
```

```
stop B stop
```

```
myData    DCD 0xC90147D2
```

MOV a register into another one

MOV <Rd>, <Rm> {, shift }

- Example: MOV r0, r1
- shift is an optional shift applied to Rm
 - ASR #n: arithmetic shift right
 - LSL #n: logical shift left
 - LSR #n : logical shift right
 - ROR #n : rotate right
 - RRX : rotate right 1 bit with sing extend
- The equivalent shift instruction is preferred:

LSL r0, r1, #3 corresponds to MOV r0, r1, LSL #3

MOV a constant into a register

MOV <Rd>, #<constant>

- The constant can be:
 - a 16-bit value (0-65535)
 - a value obtained by shifting left an 8-bit value
 - of the form 0x00XY00XY
 - of the form 0xXY00XY00
 - of the form 0xXYXYXYXY.
- MOVW is like MOV, but it takes only a 16-bit value.

Shifted constant values

Left shift	Binary	Max decimal	Max hexadecimal
0	0000000000000000000000xxxxxx	255	0xFF
2	000000000000000000000000xxxxxx00	1020	0x3FC
4	000000000000000000000000xxxxxx0000	4080	0xFF0
6	000000000000000000000000xxxxxx000000	16320	0x3FC0
8	0000000000000000xxxxxx00000000	65280	0xFF00
...
20	0000xxxxxx000000000000000000000000	$0-255 \times 2^{20}$	0xFF000000
22	00xxxxxx00000000000000000000000000	$0-255 \times 2^{22}$	0x3FC00000
24	xxxxxx0000000000000000000000000000	$0-255 \times 2^{24}$	0xFF000000

Which values are valid for MOV?

1. MOV r0, #0x00004B4B
2. MOV r0, #0x004B4B00
3. MOV r0, #0x004B0000
4. MOV r0, #0x004B004B
5. MOV r0, #0x004B4B4B
6. MOV r0, #0x4B4B0000
7. MOV r0, #0x4B000000
8. MOV r0, #0x4B4B4B4B

MVN (move negative)

- The MVN instruction moves a one's complement of the operand into a register.
- Same syntax as MOV, with one difference:
 - MVN does not accept a 16-bit value
- Example: MVN r0, #0 -> r0 = 0xFFFFFFFF
- The assembler can change a MOV into a MVN if the value is valid for MVN and not for MOV.
- MOV r0, #-2 becomes MVN r0, #1 because -2=0xFFFFFFF is not in the range of MOV.

MOVT (move top)

- MOVT moves a 16-bit value in the high halfword of a register:

```
MOVT <Rd>, #<constant>
```

- A register can be set to any 32-bit constant by using MOV and MOVT together:

```
MOV r0, #0x47D2
```

```
MOVT r0, #0xC901
```

The new value of r0 is 0xC90147D2.

LDR for loading constants

- Besides loading values from memory, LDR can be used to load constants into registers:

```
LDR <Rd>, =<constant>
```

- If constant is among the valid values of MOV, then the instruction is replaced with:

```
MOV <Rd>, #<constant>
```

- Otherwise, a block of constant, called ***literal pool***, is created and the instruction becomes:

```
LDR <Rd>, [PC, #<offset>]
```

Computation of the offset

- The offset is the difference between the address of the literal pool and PC.
- The value of PC is computed as:
 1. the address of the current instruction
 2. plus 4
 3. clearing the second bit for word alignment.
- The offset is expressed with 12 bits.

Example of offset computation

LDR r0, =0xC90147D2	0x00000118
...	...
0x47D2	0x00000144
0xC901	0x00000146

1. $0x118 = 2_000100011000$
2. $0x118 + 4 = 0x11C = 2_000100011100$
3. $PC = 2_000100011100 = 0x11C$
4. $offset = 0x144 - 0x11C = 0x28 = 40$

LDR r0, [PC, #40]

Address of the literal pool

- By default, the literal pool is placed at the `END` directive, after the last instruction.
- As the offset is 12 bits, the distance between the current instruction and the last one should be lower than 4096.
- Otherwise, the `LTORG` directive must be used to put the literal pool somewhere else.

What is the error?

```
AREA | .text|, CODE, READONLY
Reset_Handler    PROC
    EXPORT Reset_Handler [WEAK]
    LDR r0, =0xC90147D2
stop B stop
myEmptySpace    SPACE 4100
ENDP
END ;literal pool is saved here
```

Correct version

```
AREA | .text|, CODE, READONLY  
Reset_Handler    PROC  
    EXPORT Reset_Handler [WEAK]  
    LDR r0, =0xC90147D2  
    stop B stop  
    LTORG ;literal pool is saved here  
myEmptySpace    SPACE 4100  
ENDP  
END
```

Loading addresses into registers

- Two pseudo-instructions are available:

LDR <Rd>, =<label>

ADR <Rd>, <label>

- LDR creates a constant in a literal pool and uses a PC relative load to get the data.
- ADR adds or subtracts an offset to/from PC.
- ADR does not increase the code size, but it can not create all offsets
 - Addresses generated with ADR must be multiple of 4
 - ADR loads addresses in the same section.

LDR an address into a register

```
Stack_Size      EQU      0x00000200
                AREA STACK, NOINIT, READWRITE
Stack_Mem       SPACE    Stack_Size
                AREA | .text |, CODE, READONLY
...
LDR r12, =Stack_Mem
...
END ;literal pool is saved here
```

LDR an address into a register

- LDR can reference a label outside of the current section.
- In the previous example, r12 is loaded with the address of the bottom of the stack

$$r12 = r13 - 0x00000200$$

ADR an address into a register

```
AREA | .text|, CODE, READONLY
Reset_Handler    PROC
    EXPORT Reset_Handler [WEAK]
    ADR r0, myData
stop B stop
myData DCD 0xC90147D2
myEmptySpace    SPACE 4100
ENDP
```

ADR and ADRL

- The ADR pseudo-instruction is replaced with
 $LDR \langle R_d \rangle, [PC, \#<\text{offset}>]$
- The offset is expressed with 12 bits.
- If the offset is higher than 4095 bytes, ADRL must be used instead of ADR.
- ADRL generates two operations and its offset can be up to 1 MB.
- ADR and ADRL load addresses in the same section.

ADRL an address into a register

```
AREA | .text|, CODE, READONLY
Reset_Handler    PROC
    EXPORT Reset_Handler [WEAK]
    ADRL r0, myData
stop B stop
myEmptySpace    SPACE 4100
myData DCD 0xC90147D2
ENDP
```