Constants and literal pools

R. Ferrero

Politecnico di Torino

Dipartimento di Automatica e Informatica (DAUIN)

Torino - Italy

MOV

- It assigns a value to a register.
- The value can be:
  - the content of another register
  - a constant value.
- The value can not be: LDR for Address
  - an address (neither code or data address)
  - the content of a memory cell.

MOV: examples

```
     AREA myData, DATA, READONLY
myVar     DCD 0xC90147D2
     AREA |.text|, CODE, READONLY
Reset_Handler PROC
          EXPORT Reset_Handler [WEAK]
myCode    MOV r0, #15  ✓
          MOV r1, r0   ✓
          MOV r2, myCode  ✗  ;code address
          MOV r3, myVar   ✗  ;data address
          MOV r4, [myVar] ✗  ;memory content
stop      B stop
          ENDP
```

MOV a register into another one

{} means alt

```
MOV <Rd>, <Rn> {, 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 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>
```

- `constant` can be:
  - a value obtained by shifting left an 8-bit value up to 24 positions
  - of the form 0x00XY00XY
  - of the form 0xXY00XY00
  - of the form 0xXYXYXYXY.

Values of shifted 8-bit constants

Left shift	Binary	Max decimal	Max hexadecimal
0	000000000000000000000000xxxxxxxx	255	0xFF
2	0000000000000000000000xxxxxxxx00	1020	0x3FC
4	00000000000000000000xxxxxxxx0000	4080	0xFF0
6	000000000000000000xxxxxxxx000000	16320	0x3FC0
8	0000000000000000xxxxxxxx00000000	65280	0xFF00
…	…	…	…
20	0000xxxxxxxx00000000000000000000	255×2^{20}	0xFF00000
22	00xxxxxxxx0000000000000000000000	255×2^{22}	0x3FC00000
24	xxxxxxxx000000000000000000000000	255×2^{24}	0xFF000000

MVN (move negative)

- The `MVN` instruction moves the one's complement of the operand into a register.
- Same syntax as `MOV`:

```
MVN <Rd>, <Rn> {, shift}
MVN <Rd>, #<constant>
```

- Examples:

```
MVN r0, #0x14          ; r0 = 0xFFFFFFEB
MVN r1, r0, LSL #8     ; r1 = 0x000014FF
MVN r2, r0, ROR #8     ; r2 = 0x14000000
```

MOVW (move halfword)

- `MOVW` moves a 16-bit value in the low halfword of a register:

```
        MOVW <Rd>, #<constant>
```

- `constant` is restricted to a 16-bit value.
- Example:

```
MOVW r0, #0xA1B2
```

Extended use of MOV

- The assembler can replace `MOV` with `MVN` or `MOVW` if needed.
- Example 1: `MOV r0, #-2`
  It becomes `MVN r0, #1`
  Reason: `-2`= `0xFFFFFFFE` is not in the range of `MOV`.
- Example 2: `MOV r0, #0xA1B2`
  It becomes `MOVW r0, #0xA1B2`
  Reason: `0xA1B2` is a 16-bit constant.

Valid constants for MOV

- a 16-bit value: 0-65535 (`MVW`)
- a value obtained by shifting left an 8-bit constant up to 24 positions
- a value obtained by shifting left an 8-bit constant up to 24 positions, and then applying a bitwise logical NOT operation (`MVN`)
- of the form 0x00XY00XY or 0xFFXYFFXY (`MVN`)
- of the form 0xXY00XY00 or 0xXYFFXYFF (`MVN`)
- of the form 0xXYXYXYXY.

Which values are valid for MOV?

- `MOV r0, #0x00004B4B`
- `MOV r0, #0x004B4B00`
- `MOV r0, #0x004B0000`
- `MOV r0, #0x004B004B`
- `MOV r0, #0x4B4B0000`
- `MOV r0, #0xFF4B4B4B`
- `MOV r0, #0xFFFF4B00`
- `MOV r0, #0xFF4BFFFF`
- `MOV r0, #0x4BFF4BFF`
- `MOV r0, #0x4B000000`
- `MOV r0, #0x4B4B4B4B`

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 assembler puts literal pools in word-aligned addresses for faster access.

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]
```

Example of offset computation

`LDR r0, =0xC90147D2`	0x00000116
…	…
`0x47D2`	0x00000144
`0xC901`	0x00000146

1. 0x116 = 2_000100010110
2. 0x116 + 4 = 0x11A = 2_000100011010
3. `PC` = 2_000100011000 = 0x118
4. offset = 0x144 – 0x118 = 0x2C = 44

```
LDR r0, [PC, #44]
```

Address of the literal pool

- By default, the literal pool is placed at the `END` directive, after the last instruction.
- `LDR` is converted in a Thumb instruction; the offset size is 8 bits, ranging in [0, 1020].
- If the offset between the current instruction and the last one is higher, `LDR.W` can be used.
- `LDR.W` is converted in a Thumb-2 instruction; the offset size is 12 bits, ranging in [-4095, +4095].
- If the offset is still higher, the `LTORG` directive must be used to put literal pool somewhere else.

Valid Thumb instruction

```
    AREA |.text|, CODE, READONLY
Reset_Handler   PROC
    EXPORT  Reset_Handler [WEAK]
    LDR r0, =0xC90147D2
    ;becomes LDR r0, [pc, #1020]
stop B stop
myEmptySpace SPACE 1020
    ENDP
    END ;literal pool is saved here
```

Invalid Thumb instruction

```
    AREA |.text|, CODE, READONLY
Reset_Handler   PROC
    EXPORT  Reset_Handler [WEAK]
    LDR r0, =0xC90147D2
    ;error: offset out of range
stop B stop
myEmptySpace SPACE 1021
    ENDP
    END ;literal pool is saved here
```

Valid Thumb-2 instruction

```
    AREA |.text|, CODE, READONLY
Reset_Handler   PROC
    EXPORT  Reset_Handler [WEAK]
    LDR.W r0, =0xC90147D2
    ;becomes LDR.W r0, [pc, #1024]
stop B stop
myEmptySpace SPACE 1021
    ENDP
    END ;literal pool is saved here
```

Invalid Thumb-2 instruction

```
    AREA |.text|, CODE, READONLY
Reset_Handler   PROC
    EXPORT  Reset_Handler [WEAK]
    LDR.W r0, =0xC90147D2
    ;error: offset out of range
stop B stop
myEmptySpace SPACE 4095
    ENDP
    END ;literal pool is saved here
```

Valid Thumb instruction

```
    AREA |.text|, CODE, READONLY
Reset_Handler   PROC
    EXPORT  Reset_Handler [WEAK]
    LDR r0, =0xC90147D2
    ;becomes LDR r0, [pc, #0]
stop B stop
    LTORG ;literal pool is saved here
myEmptySpace SPACE 4095
    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.

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 <Rd>, [PC, #<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
```