# ADR pseudo-op

- Cannot refer to an address directly in an instruction.
- Generate value by performing arithmetic on PC.
- ADR pseudo-op generates instruction required to calculate address:

```
ADR r1,FOO
```

# ADR: loading an 32-bit address

- ADR  Rd, label
- ADRL Rd, label : long offset
- label is a PC-relative expression that evaluates to:
  - a non word-aligned address within ±255 bytes
  - a word-aligned address within ±1020 bytes.
- Use the ADRL pseudo-instruction to assemble a wider range of effective addresses.
- For pc-relative expressions, the given range is relative to a point two words after the address of the current instruction.

# Pseudo Instruction (macro)

- Example 1

```
start MOV r0,#10
      ADR r4,start ; => SUB r4,pc,#0xc
```

# ARM Assembler Directives

- Memory initialization directives

```
DCB (=) 1 byte
DCW     2 bytes
DCD (&) 4 bytes        : aligned to 4 bytes
DCQ     8 bytes
DCI     2 or 4 bytes : defining instruction
```

- END : end of a source file
- EQU : #define in C

# Example

```
a       SETA        34906
addr DCD            0xA10E
        LDR r4,=&1000000F
        DCD         2_11001010
c3      SETA        8_74007
        DCQ         0x0123456789abcdef
        LDR r1,='A'      ; pseudo-inst.loading 65 into r1
        ADD r3,r2,#'□' ; add 39 to contents of r2
a       SETA        256*256          ; a numeric expression
        MOV r1,#(a*22) ; (a*22) is another expression
```

# ARM Assembler Directives

- A: unsigned 32-bit integer
- S: ACSII string
- L: logical
- Decalre globally: GBLA, GBLS, GLBL
- Declare locally to a macro: LCLA, LCLS, LCLL
- Set value: SETA, SETS, SETL

# ARM Assembler Directives

- MAP (^), FIELD(#): defining objects

```
        MAP 0 ; base address
count   FIELD 4       ; define an int called count
type    FIELD 4       ; define an int called type
size    FIELD 0       ;record the struct size

        SUB   sp, sp #size   ; mae room on the stack
        MOV   r0, #0
        STR   r0, [sp, #count]
        STR   r0, [sp, #type]
```

# Example: C assignments

- C:

```
x = (a + b) - c;
```

- Assembler

```
AREA CODE      ; beginning of memory block for program
ENTRY          ; program start at the following
ADR r4,a       ; get address for a
LDR r0,[r4]    ; get value of a
ADR r4,b       ; get address for b, reusing r4
ADR r1,[r4]    ; get value of b
ADD r3,r0,r1; compute a+b
ADR r4,c       ; get address for c
LDR r2[r4]     ; get value of c
```

# C assignment, cont'd.

```
    SUB r3,r3,r2; complete computation of x
    ADR r4,x    ; get address for x
    STR r3[r4]  ; store value of x
    AREA DATA   ; beginning of memory block for data
a   DCD 0       ; a initialized to 0
b   DCD 0       ; b initialized to 0
c   DCD 0       ; c initialized to 0
x   DCD 0       ; x initialized to 0
```

# LDR: loading a 32-bit constant

- Although the MOV/MVN mechanism will load a large range of constants into a register, sometimes this mechanism will not generate the required constant.
- Therefore, the assembler also provides a method which will load *ANY* 32 bit constant:
  - `LDR rd,=numeric_constant`

# LDR: loading 32-bit constants

- If the constant can be constructed using either a MOV or MVN then this will be the instruction actually generated.
- Otherwise, the assembler will produce an LDR instruction with a PC-relative address to read the constant from a literal pool.
  - `LDR r0,=0x42`

    `; MOV r0,#0x42`
  - `LDR r0,=0x55555555`

    `; LDR r0,[pc, offset_to_literal]`

# LDR: loading 32-bit constants

- `LDR r0,=0xff00ffff`
  `; MVN r0,#0x00ff0000`
- LDR Rd, =constant

```
        ADR Rd, address
        …

address DCD constant
```

- As this mechanism will always generate the best instruction for a given case, it is the recommended way of loading constants.

# Pseudo Instruction: ADR

- ADR always assembles to one instruction. The assembler attempts to produce a single ADD or SUB instruction to load the address. If the address cannot be constructed in a single instruction, an error is generated and the assembly fails.

# Example: C assignment

- C:

```
z = (a << 2) |  (b & 15);
```

- Assembler:

```
ADR r4,a          ; get address for a
LDR r0,[r4]       ; get value of a
MOV r0,r0,LSL 2 ; perform shift
ADR r4,b          ; get address for b
LDR r1,[r4]       ; get value of b
AND r1,r1,#15     ; perform AND
ORR r1,r0,r1      ; perform OR
ADR r4,z          ; get address for z
STR r1,[r4]       ; store value for z
```

# Example: if statement

- C:

```
if (a > b) { x = 5; y = c + d; } else x = c - d;
```

- Assembler:

```
; compute and test condition
  ADR r4,a ; get address for a
  LDR r0,[r4] ; get value of a
  ADR r4,b ; get address for b
  LDR r1,[r4] ; get value for b
  CMP r0,r1 ; compare a < b
  BGE fblock ; if a >= b, branch to false block
```

# If statement, cont'd.

```
; true block
   MOV r0,#5 ; generate value for x
   ADR r4,x ; get address for x
   STR r0,[r4] ; store x
   ADR r4,c ; get address for c
   LDR r0,[r4] ; get value of c
   ADR r4,d ; get address for d
   LDR r1,[r4] ; get value of d
   ADD r0,r0,r1 ; compute y
   ADR r4,y ; get address for y
   STR r0,[r4] ; store y
   B after ; branch around false block
```

# If statement, cont'd.

```
; false block
fblock ADR r4,c ; get address for c
   LDR r0,[r4] ; get value of c
   ADR r4,d ; get address for d
   LDR r1,[r4] ; get value for d
   SUB r0,r0,r1 ; compute a-b
   ADR r4,x ; get address for x
   STR r0,[r4] ; store value of x
after ...
```

# Example: Conditional instruction implementation

```
; true block
   MOVLT r0,#5 ; generate value for x
   ADRLT r4,x ; get address for x
   STRLT r0,[r4] ; store x
   ADRLT r4,c ; get address for c
   LDRLT r0,[r4] ; get value of c
   ADRLT r4,d ; get address for d
   LDRLT r1,[r4] ; get value of d
   ADDLT r0,r0,r1 ; compute y
   ADRLT r4,y ; get address for y
   STRLT r0,[r4] ; store y
```

# Example: switch statement

- C:

```
switch (test) { case 0: … break; case 1: … }
```

- Assembler:

```
  ADR r2,test ; get address for test
  LDR r0,[r2] ; load value for test
  ADR r1,switchtab ; load address for switch table
  LDR r1,[r1,r0,LSL #2] ; index switch table
switchtab DCD case0
  DCD case1
...
```

# Example: FIR filter

- C:

```
for (i=0, f=0; i<N; i++)
  f = f + c[i]*x[i];
```

- Assembler

```
; loop initiation code
  MOV r0,#0 ; use r0 for I
  MOV r8,#0 ; use separate index for arrays
  ADR r2,N ; get address for N
  LDR r1,[r2] ; get value of N
  MOV r2,#0 ; use r2 for f
```

# FIR filter, cont'.d

```
   ADR r3,c ; load r3 with base of c
   ADR r5,x ; load r5 with base of x
; loop body
loop LDR r4,[r3,r8] ; get c[i]
   LDR r6,[r5,r8] ; get x[i]
   MUL r4,r4,r6 ; compute c[i]*x[i]
   ADD r2,r2,r4 ; add into running sum
   ADD r8,r8,#4 ; add one word offset to array index
   ADD r0,r0,#1 ; add 1 to i
   CMP r0,r1 ; exit?
   BLT loop ; if i < N, continue
```