

Systems Architecture

ARM Assembler

Data Movement



Memory Access

Load Register from memory

Store Register in memory

```
\begin{array}{lll} \mathtt{STR}\langle cc\rangle & \mathtt{R}s \text{, } \langle op2\rangle & \langle cc\rangle \mathtt{:} \ \mathsf{MAR} & \leftarrow \langle op2\rangle \\ & \langle cc\rangle \mathtt{:} \ \mathsf{MBR} & \leftarrow \mathtt{R}s \\ & \langle cc\rangle \mathtt{:} \ \mathsf{M}(\mathsf{MAR}) \leftarrow \mathsf{MBR} \end{array}
```

 Memory Reference must be 32-bit word aligned otherwise a Data Abort Exception will occur use the ALIGN directive to force alignment



Load / Store Byte

Load Register with unsigned Byte from memory

```
\begin{array}{lll} \operatorname{LDR}\langle cc \rangle \operatorname{B} & \operatorname{R} d \text{ , } \langle op2 \rangle & \langle cc \rangle \colon \operatorname{MAR} & \leftarrow \langle op2 \rangle \\ & \langle cc \rangle \colon \operatorname{MBR} & \leftarrow \operatorname{M(MAR)} \\ & \langle cc \rangle \colon \operatorname{R} d (7:0) & \leftarrow \operatorname{MBR} \\ & \langle cc \rangle \colon \operatorname{R} d (31:8) \leftarrow 0 \end{array}
```

Load Register with Signed Byte from memory

```
\begin{array}{lll} \text{LDR}\langle cc\rangle \text{SB} & \text{R}d \text{ , } \langle op2\rangle & \langle cc\rangle \text{: MAR} & \leftarrow \langle op2\rangle \\ & \langle cc\rangle \text{: MBR} & \leftarrow \text{M(MAR)} \\ & \langle cc\rangle \text{: R}d(7\text{:}0) & \leftarrow \text{MBR} \\ & \langle cc\rangle \text{: R}d(31\text{:}8) \leftarrow \text{R}d(7) \end{array}
```

Store Register in a Byte of memory

```
\begin{array}{lll} \mathtt{STR}\langle cc \rangle \mathtt{B} & \mathtt{R}s \text{, } \langle op2 \rangle & \langle cc \rangle \mathtt{:} \ \mathsf{MAR} & \leftarrow \langle op2 \rangle \\ & \langle cc \rangle \mathtt{:} \ \mathsf{MBR} & \leftarrow \mathtt{R}s \\ & \langle cc \rangle \mathtt{:} \ \mathsf{M}(\mathsf{MAR}) \leftarrow \mathtt{R}s(\mathsf{7}\mathtt{:}\mathsf{0}) \end{array}
```



Load / Store Halfword

- Does not work in the ARMulator
- An ARM word is 32-bits, so a Halfword is 16-bits
- Memory Reference must be Halfword aligned
- Load Register with unsigned Halfword from memory

Load Register with Signed Halfword from memory

```
 \begin{array}{lll} \text{LDR}\langle cc \rangle \text{SH} & \text{R}d \text{ , } \langle op2 \rangle & \langle cc \rangle \text{: MAR} & \leftarrow \langle op2 \rangle \\ & \langle cc \rangle \text{: MBR} & \leftarrow \text{M(MAR)} \\ & \langle cc \rangle \text{: R}d \text{(15:0)} & \leftarrow \text{MBR} \\ & \langle cc \rangle \text{: R}d \text{(31:16)} \leftarrow \text{R}d \text{(15)} \\ \end{array}
```

Store Register in a Halfword of memory

```
STR\langle cc\rangle H \qquad Rs \,, \ \langle op2\rangle \qquad \langle cc\rangle : MAR \qquad \leftarrow \langle op2\rangle \\ \langle cc\rangle : MBR \qquad \leftarrow Rs \\ \langle cc\rangle : M(MAR) \qquad \leftarrow MBR(15:0)
```



```
; 16bit data transfer
 2.
 3.
                       move16 – 16-bit data transfer
             TTL
 4.
             AREA
                       Program, CODE, READONLY
 5.
              ENTRY
 6.
 7.
     Main
 8.
             LDRB
                       R1, Value
                                   ; Load value
 9.
             STR
                       R1, Result
                                    ; Sore it again
10.
              SWI
                       &11
                                    ; exit()
11.
12.
                       &C123
             DCW
     Value
                                    ; Source value to be moved
13.
             ALIGN
                                    ; Alling next word
             DCW
14.
                                    ; Reserve space for result
     Result
                       0
15.
16.
              END
```



```
; 16bit data transfer
 2.
 3.
             TTL
                       move16 – 16-bit data transfer
             AREA
 4.
                       Program, CODE, READONLY
 5.
             ENTRY
 6.
 7.
     Main
 8.
             LDRB
                       R1, Value ; Load value
 9.
             STR
                       R1, Result
                                   ; Sore it again
10.
             SWI
                       &11
                                    ; exit()
11.
12.
                       &C123
             DCW
     Value
                                    ; Source value to be moved
13.
             ALIGN
                                    ; Alling next word
                                    ; Reserve space for result
             DCW
14.
     Result
                       0
15.
16.
             END
             Define Program Title
```



```
; 16bit data transfer
    2.
    3.
                          move16 – 16-bit data transfer
                TTL
                AREA
                          Program, CODE, READONLY
    4.
    5.
                ENTRY
    6.
    7.
        Main
    8.
                LDRB
                          R1, Value ; Load value
    9.
                STR
                          R1, Result
                                      ; Sore it again
   10.
                SWI
                          &11
                                      ; exit()
   11.
   12.
                          &C123
        Value
                DCW
                                      ; Source value to be moved
                ALIGN
                                      ; Alling next word
   13.
                                      ; Reserve space for result
   14.
        Result
                DCW
                          0
   15.
                END
   16.
AREA
                Label Program Area
                Code or Data space; Read Only or Read / Write
```



```
; 16bit data transfer
    2.
    3.
                          move16 – 16-bit data transfer
                TTL
                AREA
                          Program, CODE, READONLY
                ENTRY
    5.
    6.
    7.
        Main
    8.
                LDRB
                          R1, Value ; Load value
    9.
                STR
                          R1, Result
                                      ; Sore it again
   10.
                SWI
                          &11
                                       ; exit()
   11.
   12.
                          &C123
        Value
                DCW
                                       ; Source value to be moved
                ALIGN
                                       ; Alling next word
   13.
                DCW
                                       ; Reserve space for result
   14.
        Result
                          0
   15.
   16.
                END
ENTRY
                Define Program Entry Point
```



```
; 16bit data transfer
    2.
    3.
                          move16 – 16-bit data transfer
                TTL
    4.
                AREA
                          Program, CODE, READONLY
    5.
                 ENTRY
    6.
    7.
        Main
    8.
                LDRB
                          R1, Value ; Load value
    9.
                STR
                          R1, Result
                                      ; Sore it again
   10.
                SWI
                          &11
                                       ; exit()
   11.
                          &C123
   12.
        Value
                DCW
                                       ; Source value to be moved
                ALIGN
                                       ; Alling next word
   13.
                                       ; Reserve space for result
   14.
        Result
                DCW
                          0
   15.
   16.
                 END
Main
                Label the memory address
                Debug will place breakpoint at Main
```



```
; 16bit data transfer
    2.
    3.
                          move16 – 16-bit data transfer
                TTL
    4.
                AREA
                          Program, CODE, READONLY
    5.
                 ENTRY
    6.
    7.
        Main
    8.
                 LDRB
                          R1, Value
                                      ; Load value
    9.
                 STR
                          R1, Result
                                       ; Sore it again
   10.
                 SWI
                          &11
                                       ; exit()
   11.
   12.
                          &C123
        Value
                DCW
                                       ; Source value to be moved
                ALIGN
                                       ; Alling next word
   13.
                DCW
                                       ; Reserve space for result
   14.
        Result
                          0
   15.
   16.
                 END
SWI
                Software Interrupt — Call the Operating System
                exit()
```



&

```
; 16bit data transfer
 2.
 3.
                       move16 – 16-bit data transfer
             TTL
 4.
             AREA
                       Program, CODE, READONLY
 5.
              ENTRY
 6.
 7.
     Main
 8.
             LDRB
                       R1, Value
                                   ; Load value
 9.
             STR
                       R1, Result
                                    ; Sore it again
10.
              SWI
                       &11
                                    ; exit()
11.
                       &C123
12.
             DCW
     Value
                                    ; Source value to be moved
13.
             ALIGN
                                    ; Alling next word
                                    ; Reserve space for result
             DCW
14.
     Result
                       0
15.
16.
              END
             Define a Hexadecimal value
```



DCW

Program: move16.s

```
; 16bit data transfer
 2.
 3.
                       move16 – 16-bit data transfer
             TTL
 4.
             AREA
                       Program, CODE, READONLY
 5.
              ENTRY
 6.
 7.
     Main
 8.
             LDRB
                       R1, Value ; Load value
 9.
             STR
                       R1, Result
                                   ; Sore it again
10.
             SWI
                       &11
                                    ; exit()
11.
12.
                       &C123
             DCW
     Value
                                    ; Source value to be moved
13.
             ALIGN
                                    ; Alling next word
             DCW
14.
                                    ; Reserve space for result
     Result
                       0
15.
16.
              END
```

Define a 16-bit data value



```
; 16bit data transfer
    2.
    3.
                          move16 – 16-bit data transfer
                 TTL
    4.
                 AREA
                          Program, CODE, READONLY
    5.
                 ENTRY
    6.
    7.
        Main
    8.
                 LDRB
                          R1, Value ; Load value
    9.
                 STR
                          R1, Result
                                      ; Sore it again
   10.
                 SWI
                          &11
                                       ; exit()
   11.
                          &C123
   12.
        Value
                 DCW
                                       ; Source value to be moved
   13.
                ALIGN
                                       ; Alling next word
   14.
                 DCW
                                       ; Reserve space for result
        Result
                          0
   15.
   16.
                 END
ALIGN
                Align data item on 32-bit word boundary
```



```
; 16bit data transfer
    2.
    3.
                          move16 – 16-bit data transfer
                TTL
    4.
                AREA
                          Program, CODE, READONLY
    5.
                ENTRY
    6.
    7.
        Main
    8.
                LDRB
                          R1, Value ; Load value
    9.
                STR
                          R1, Result
                                      ; Sore it again
   10.
                SWI
                          &11
                                      ; exit()
   11.
   12.
                          &C123
        Value
                DCW
                                      ; Source value to be moved
                ALIGN
                                      ; Alling next word
   13.
        Result
                DCW
                                      ; Reserve space for result
   14.
                          0
   15.
   16.
                END
END
```

End of program source



```
; 16bit data transfer
    2.
    3.
                          move16 – 16-bit data transfer
                 TTL
    4.
                 AREA
                           Program, CODE, READONLY
    5.
                 ENTRY
    6.
    7.
        Main
    8.
                 LDRB
                           R1, Value
                                      ; Load value
    9.
                 STR
                           R1, Result
                                       ; Sore it again
   10.
                 SWI
                           &11
                                       ; exit()
   11.
                          &C123
   12.
        Value
                 DCW
                                       ; Source value to be moved
                 ALIGN
                                       ; Alling next word
   13.
                 DCW
                                       ; Reserve space for result
   14.
        Result
                           0
   15.
   16.
                 END
Bug
                 Assembler can only find syntax errors
                 You have to find the logical errors
```



Data Movement

MOV
 Move Data

MOVS Move Data and Set Zero and Negative flags

 $MOV\langle cc \rangle$ Move Data if $\langle cc \rangle$

• $MOV\langle cc\rangle\langle S\rangle$ Rd, $\langle op1\rangle$ $\langle cc\rangle$: ALU $\leftarrow \langle op1\rangle$

 $\langle cc \rangle$: Rd \leftarrow ALU

 $\langle S \rangle \langle cc \rangle$: CPSR \leftarrow ALU(Flags)

Move and Negate Data

 $MVN\langle cc\rangle\langle S\rangle$ Rd, $\langle op1\rangle$ $\langle cc\rangle$

 $\langle cc \rangle$: ALU $\leftarrow \langle op1 \rangle$

 $\langle cc \rangle$: Rd \leftarrow ALU

 $\langle S \rangle \langle cc \rangle$: CPSR \leftarrow ALU(Flags)

- Rd is the destination (must be a register)
- (op1) is the source



```
; Find the one's compliment (inverse) of a number
 2.
3.
                       invert.s – one's complement
             TTL
4.
                       Program, CODE, READONLY
             AREA
5.
             ENTRY
6.
 7.
     Main
8.
             LDR
                       R1, Value
                                    ; Load number to be processed
9.
                       R1, R1
             MVN
                                    ; Invert (not) the value
10.
             STR
                                    ; Store the result
                       R1, Result
11.
             SWI
                       &11
                                    ; exit()
12.
13.
     Value
             DCD
                       &C123
                                    ; Value to be complemented
14.
                                    ; Reserve space for result
     Result
             DCD
                       0
15.
16.
             END
```



```
; Find the one's compliment (inverse) of a number
   2.
   3.
                          invert.s – one's complement
                TTL
   4.
                AREA
                          Program, CODE, READONLY
   5.
                ENTRY
   6.
   7.
       Main
   8.
                LDR
                          R1, Value
                                      ; Load number to be processed
   9.
                MVN
                          R1, R1
                                      ; Invert (not) the value
  10.
                STR
                          R1, Result
                                      ; Store the result
  11.
                SWI
                          &11
                                      ; exit()
  12.
                          &C123
  13.
       Value
                DCD
                                      ; Value to be complemented
  14.
                                      ; Reserve space for result
       Result
                DCD
                          0
  15.
  16.
                END
Labels
                Used to access memory directly
```



```
; Find the one's compliment (inverse) of a number
   2.
   3.
                          invert.s – one's complement
                TTL
   4.
                AREA
                          Program, CODE, READONLY
   5.
                ENTRY
   6.
   7.
       Main
   8.
                LDR
                          R1, Value
                                      ; Load number to be processed
   9.
                MVN
                          R1, R1
                                      ; Invert (not) the value
  10.
                STR
                          R1, Result
                                      ; Store the result
  11.
                SWI
                          &11
                                      ; exit()
  12.
  13.
       Value
                DCD
                          &C123
                                      ; Value to be complemented
                                      ; Reserve space for result
  14.
       Result
                DCD
                          0
  15.
  16.
                END
                Used to define (and initialise) memory values
DCD
                No need for ALIGN as DCD defines 32-bit values
```



```
; Find the one's compliment (inverse) of a number
   2.
   3.
                          invert.s – one's complement
                TTL
   4.
                          Program, CODE, READONLY
                AREA
   5.
                ENTRY
   6.
   7.
       Main
   8.
                LDR
                          R1, Value
                                       ; Load number to be processed
   9.
                MVN
                          R1, R1
                                       ; Invert (not) the value
                          R1, Result
  10.
                STR
                                       ; Store the result
  11.
                SWI
                          &11
                                       ; exit()
  12.
                          &C123
  13.
        Value
                DCD
                                       ; Value to be complemented
  14.
                                       ; Reserve space for result
        Result
                DCD
                          0
  15.
  16.
                END
MNV
                 Move and Negate
                 Uses same register for Source<sub>1</sub> and Destination
```



Arithmetic

Addition

$$\begin{array}{lll} \text{ADD}\langle cc\rangle\langle S\rangle & \text{R}d \text{ , } \text{R}n \text{ , } \langle op1\rangle & \langle cc\rangle \text{: ALU } \leftarrow \text{R}n + \langle op1\rangle \\ & \langle cc\rangle \text{: R}d & \leftarrow \text{ALU} \\ & \langle S\rangle\langle cc\rangle \text{: CPSR} \leftarrow \text{ALU(Flags)} \end{array}$$

Subtraction

$$\begin{array}{lll} \mathtt{SUB}\langle cc\rangle\langle S\rangle & \mathtt{R}d \text{ , } \mathtt{R}n \text{ , } \langle op1\rangle & \langle cc\rangle \mathtt{: ALU} & \leftarrow \mathtt{R}n - \langle op1\rangle \\ & \langle cc\rangle \mathtt{: R}d & \leftarrow \mathtt{ALU} \\ & \langle S\rangle\langle cc\rangle \mathtt{: CPSR} \leftarrow \mathtt{ALU(Flags)} \end{array}$$

Multiplication

Multiply two 16-bit values (Rn and Rs) producing a 32-bit result (Rd)

Division
 There is **no** division instruction



: Add two numbers and store the result Main 7. 8. LDR R0, =Value1 ; R0 = &Value1 9. LDR R1, [R0] ; R1 = *R010. ADD R0, R0, #0x4 ; R0++ 11. ; R2 = *R0LDR R2, [R0] 12. ; R1 = R1 + R2ADD R1, R1, R2 13. LDR R0, =Result ; R0 = &Result14. STR R1, [R0] ; *R0 = R115. SWI &11 ; exit(0) 16. Value1 DCD &37E3C123 17. DCD &367402AA 18. Value2 DCD 0 19. Result



; Add two numbers and store the result

```
Main
8.
           LDR
                R0, =Value1 ; R0 = &Value1
9.
           LDR R1, [R0] ; R1 = *R0
```

10. R0, R0, #0x4 ; R0++ ADD

11. LDR R2, [R0] ; R2 = *R0

12. ADD R1, R1, R2 ; R1 = R1 + R2

13. LDR R0, =Result ; R0 = &Result

14. STR R1, [R0] ; *R0 = R1

15. SWI &11 ; exit(0)

16.

17. Value1 DCD &37E3C123

18. Value2 DCD &367402AA

DCD 0 19. Result

Lines of no interest are ignored



; Add two numbers and store the result Main 7. 8. R0, =Value1 ; R0 = &Value1 LDR 9. LDR R1, [R0]; R1 = *R010. R0, R0, #0x4 ; R0++ ADD 11. LDR R2, [R0] ; R2 = *R012. ADD R1, R1, R2 ; R1 = R1 + R213. LDR R0, =Result ; R0 = &Result14. STR R1, [R0] ; *R0 = R115. SWI &11 ; exit(0) 16. 17. Value1 DCD &37E3C123 18. Value2 DCD &367402AA DCD 0 19. Result

ADD Same register for Source₁ and Destination



=label

Program: add2.s

; Add two numbers and store the result Main 8. R0, =Value1 ; R0 = &Value1 LDR 9. R1, [R0] ; R1 = *R0LDR 10. R0, R0, #0x4 ; R0++ ADD 11. LDR R2, [R0] ; R2 = *R012. : R1 = R1 + R2ADD R1, R1, R2 13. R0, =Result ; R0 = &ResultLDR 14. STR R1, [R0] ; *R0 = R115. SWI &11 ; exit(0) 16. 17. Value1 DCD &37E3C123 DCD &367402AA 18. Value2 DCD 0 19. Result

Load address of *label* into R0

Beginning Programs - p. 9/10



```
; Add two numbers and store the result
    Main
7.
8.
                  R0, =Value1
                               ; R0 = &Value1
            LDR
9.
            LDR
                  R1, [R0]
                                ; R1 = *R0
10.
                   R0, R0, #0x4
                                ; R0++
            ADD
                                ; R2 = *R0
11.
                  R2, [R0]
            LDR
12.
            ADD
                  R1, R1, R2
                                ; R1 = R1 + R2
13.
                               ; R0 = &Result
            LDR R0, =Result
14.
            STR R1, [R0]
                                ; *R0 = R1
15.
            SWI
                  &11
                                ; exit(0)
16.
17. Value1
            DCD &37E3C123
18. Value2
            DCD &367402AA
            DCD 0
19. Result
```

LDR Load data from memory pointed to by R0



```
; Add two numbers and store the result
    Main
7.
8.
                  R0, =Value1
                              ; R0 = &Value1
            LDR
9.
            LDR
                  R1, [R0]
                          ; R1 = *R0
10.
                  R0, R0, #0x4 ; R0++
            ADD
                               : R2 = *R0
11.
            LDR
                  R2, [R0]
12.
            ADD
                  R1, R1, R2 ; R1 = R1 + R2
13.
            LDR R0, =Result
                              ; R0 = &Result
14.
            STR
                  R1, [R0]
                               ; *R0 = R1
15.
            SWI
                  &11
                               ; exit(0)
16.
17. Value1
            DCD &37E3C123
            DCD &367402AA
18. Value2
            DCD 0
19. Result
```

ADD Increment pointer in R0 by a word (4 bytes)



```
; Add two numbers and store the result
    Main
7.
8.
                  R0, =Value1 ; R0 = &Value1
            LDR
                  R1, [R0] ; R1 = *R0
9.
            LDR
10.
                  R0, R0, #0x4
                               ; R0++
            ADD
                               : R2 = *R0
11.
                  R2, [R0]
            LDR
                  R1, R1, R2 ; R1 = R1 + R2
12.
            ADD
                               ; R0 = \&Result
13.
            LDR
                  R0, =Result
14.
                  R1, [R0]
                               ; *R0 = R1
            STR
15.
            SWI
                  &11
                               ; exit(0)
16.
17.
   Value1
          DCD &37E3C123
18.
   Value2 DCD &367402AA
          DCD
19. Result
```

ADD/LDR No need for ADD instructions if LDR uses post-index addressing: [R0], #0x4 or *(R0++) in C



; Add two numbers and store the result Main 7. 8. R0, =Value1 ; R0 = &Value1 LDR 9. LDR R1, [R0]; R1 = *R010. R0, R0, #0x4 ; R0++ ADD 11. LDR R2, [R0] ; R2 = *R012. ADD R1, R1, R2 ; R1 = R1 + R213. LDR R0, =Result ; R0 = &Result14. STR R1, [R0] ; *R0 = R115. SWI &11 ; exit(0) 16. DCD &37E3C123 17. Value1 DCD &367402AA 18. Value2 DCD 0 19. Result

Store data indirect (at memory pointed to by R0)



```
; Add two numbers and store the result
    Main
7.
8.
                  R0, =Value1 ; R0 = &Value1
            LDR
9.
            LDR
                  R1, [R0]; R1 = *R0
10.
                  R0, R0, #0x4 ; R0++
            ADD
11.
            LDR R2, [R0] ; R2 = *R0
12.
            ADD
                  R1, R1, R2 ; R1 = R1 + R2
13.
            LDR R0, =Result ; R0 = \&Result
14.
            STR R1, [R0]
                               ; *R0 = R1
15.
            SWI
                  &11
                               ; exit(0)
16.
17. Value1
            DCD &37E3C123
18. Value2
            DCD &367402AA
            DCD 0
19. Result
```

Comments These are *bad* comments

Comments should say why not what



Program: shiftleft.s

```
; Shift Left one bit
  2.
  3.
                         shiftleft.s
               TTL
  4.
               AREA
                          Program, CODE, READONLY
  5.
               ENTRY
  6.
  7.
      Main
  8.
               LDR
                          R1, Value
                                              ; Load the value to be shifted
  9.
               MOV
                          R1, R1, LSL #0x1
                                              ; Shift Left one bit
 10.
                         R1, Result
                                              ; Store the result
               STR
 11.
               SWI
                         &11
                                              ; exit
 12.
 13.
      Value
               DCD
                         &4242
                                              ; Value to be shifted
 14.
      Result
               DCD
                                              ; Space to store result
 15.
 16.
               END
LSL
                 Logical Shift Left by 1 bit
```