Example: Calculating the Absolute Value

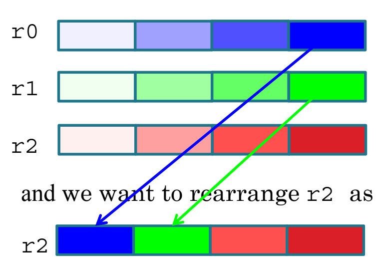
- To calculate x ← |x|, where x is a signed integer, we can implement `if x < 0 then x = -x`
- In ARM

```
TEQ     r0, #0          ;compare r0 with zero
RSBMI   r0, r0, #0      ;if negative (MInus) r0 ← 0 – r0
```

- What is the difference between `TEQ` and `CMP`?

Example: Byte Manipulation and Concatenation

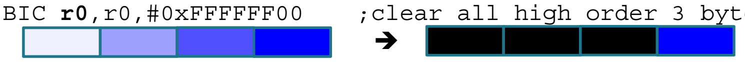
- Suppose we have `r0`, `r1`, and `r2` as follow:



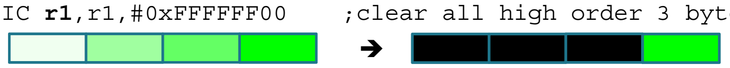

```
BIC r0,r0,#0xFFFFFF00    ;clear all high order 3 bytes
BIC r1,r1,#0xFFFFFF00    ;clear all high order 3 bytes
ADD r2,r1,r2,LSL#16      ;LSL r2 by 2 bytes & add r1 to it
ADD r2,r2,r0,LSL#8       ;LSL r0 by 1 byte & add it to r2
MOV r2,r2,ROR#16         ;Swap the two r2 16 bits together
```

Example: Byte Manipulation and Concatenation

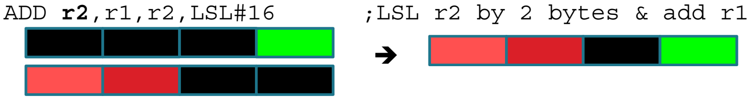
- BIC **r0**,r0,#0xFFFFFF00 ;clear all high order 3 bytes



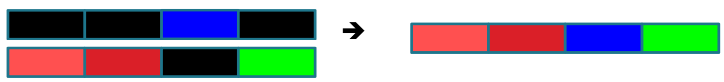
BIC **r1**,r1,#0xFFFFFF00 ;clear all high order 3 bytes



ADD **r2**,r1,r2,LSL#16 ;LSL r2 by 2 bytes & add r1 to it



ADD **r2**,r2,r0,LSL#8 ;LSL r0 by 1 byte & add it to r2



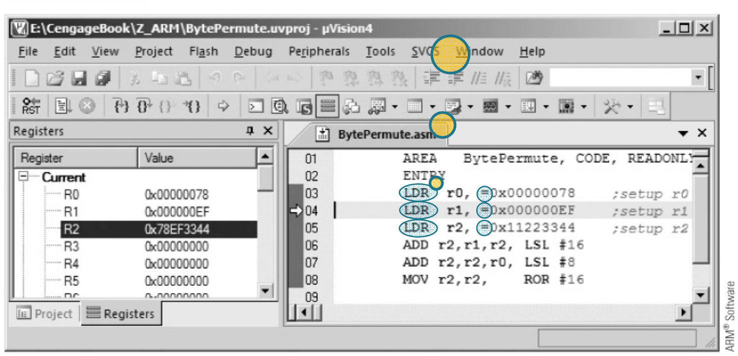
MOV **r2**,r2,ROR#16 ;Swap the two r2 16 bits together

Example: Byte Manipulation and Concatenation

LDR is a Pseudo instruction.
The assembler will uses a **MOV** or **MVN** (if it can).

O.W., **LDR ri,[pc,#offset]** instruction is used to access the constant that will be stored by the assembly in a ***literal pool***.

FIGURE 3.44 Manipulating bytes



Example: Byte Reversal (Big-endian to Little-endian)

- Suppose that **0xAB CD EF GH** is in stored in `r0`
- We want to reverse the content of `r0`, `i.e.,` store **0xGH EF CD AB** in `r0`
- We will use `r1` as a working register
- Let us revise the XOR truth table
  - $x \oplus 0 = x$
  - $x \oplus x = 0$
  - $x \oplus x \oplus y = y$

A	B	$C = A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

```
EOR r1,r0,r0, ROR#16  ;A⊕E,B⊕F,C⊕G,D⊕H,E⊕A,F⊕B,G⊕C,H⊕D
BIC r1,r1,#0x00FF0000 ;A⊕E,B⊕F,  0,  0,E⊕A,F⊕B,G⊕C,H⊕D
MOV r0,r0,ROR#8       ; G , H , A , B , C , D , E , F
EOR r0,r0,r1,LSR#8    ;r1 after LSR#8 is
                      ; 0 , 0 ,A⊕E,B⊕F, 0 , 0 ,E⊕A,F⊕B
                      ;The final result will be
                      ; G , H ,A⊕A⊕E,B⊕B⊕F,C,D,E⊕E⊕A,F⊕F⊕B
                      ; G , H , E , F , C , D , A , B
```


Example: Variable Swapping

- Assume that we have two variables stored in **r0** and **r1**
- We wants to swap these two variables
  - [r2] ← [r0]
  - [r0] ← [r1]
  - [r1] ← [r2]
- Now, we want to do the same thing without using **r2**
- Let us revise the XOR truth table
  - $x \oplus 0 = x$
  - $x \oplus x = 0$
  - $\mathbf{x \oplus x \oplus y = y}$

A	B	$C = A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

```
EOR r0,r0,r1    ; [r0] ← [r0] ⊕ [r1]
EOR r1,r0,r1    ; [r1] ← [r0] ⊕ [r1]
                ; [r1] ←([r0] ⊕ [r1]) ⊕ [r1]
                ; [r1] ← [r0]
EOR r0,r0,r1    ; [r0] ← [r0] ⊕ [r1]
                ; [r0] ←([r0] ⊕ [r1]) ⊕ [r0]
                ; [r0] ← [r1]
```

X ← X ⊕ Y
Y ← X ⊕ Y
X ← X ⊕ Y

Example: Multiplication by 2^n - 1, 2^n, or 2^n + 1

- Multiplying by 2^n can be implemented using `MOV` instruction and `LSL#n`
- Example

```
MOV r2, r1, LSL#4      ; [r2]←[r1]^4
```

- Multiplying by 2^n + 1 can be implemented using `ADD` instruction and `LSL#n`
- Example

```
ADD r2, r1, r1, LSL#4 ; [r2]←[r1] + [r1]^4
```

- Multiplying by 2^n - 1 can be implemented using `RSB` instruction and `LSL#n`
- Example

```
RSB r2, r1, r1, LSL#4 ; [r2]←[r1]^4 - [r1]
```

203

Computer Organization and Architecture: Themes and Variations, 1st Edition Clements

Example: Multiplication by 2^n - 1, 2^n, or 2^n + 1

- Let us translate the following C code

```
if(x > y)
  p = 17 * q;
else
{ if(x = y)
    p = 16 * q;
  else /* i.e., x < y */
     p= 15 * q;
}
```

- Assume that `x` and `y` are stored in `r2` and `r3`, and also that `p` and `q` are `r4` and `r1`

```
CMP   r2, r3               ;Compare x and y
ADDGT r4, r1, r1, LSL#4 ; IF >, then p ← q   + q^4
MOVEQ r4, r1, LSL#4      ; IF =, then p ← q^4
RSBLT r4, r1, r1, LSL#4 ; IF <, then p ← q^4 - q
```

`r4` not `r1`
Not correct in
the book page
200

204

Example: Converting Capital Letter ➔ Small Letter

- Let us convert any capital letter to small letter
- Capital letters begins by `'A'` and end by `'Z'`
- Assume that the character to be converted in `r0` and `r1` is a working register

```
CMP     r0, #'A'        ;Are we in the range of the capital?
RSBGES  r1, r0, #'Z'    ;If >= A,
                        ;then check with Z
                        ;      and update the flags
ORRGE   r0, r0, #0x0020;If between A and Z inclusive,
                        ;then set bit 5 to force lower case
```


Computer Organization and Architecture: Themes and Variations, 1st Edition Clements

Example: If Statement in One Instruction!!

- Let us translate the following C code

```
if(x < 0)
  x = 0;
```

- Assume that `x` is stored in `r0`

```
BIC r0, r0, r0, ASR#31 ; only one instruction!!
```

- `ASR#31` will fill all bits of `r0` with the sign bit
  - If positive, the result will be `0x00000000`
  - If negative, the result will be `0xFFFFFFFF`

  Hence, if negative, all bits will be cleared, i.e., `x` ← `0`
  Otherwise, `x` will stay as it is without change

Example: Simple Bit-level Logical Operations

- Assume #2_0000 0000 0000 0000 0000 0000 0000 **pqrs** is stored in `r0`
- We wish to implement the following statement

```
if ((p == 1) && (r == 1))
    s = 1;
```

- Assume that `r1` is a working register

```
ANDS   r1,r0,#0x8 ;clear all bits in r1 and copy p from r0
ANDNES r1,r0,#0x2 ;if p == 1,
                  ; clear all bits in r1 and copy r from r0
ORRNE  r0,r0,#1   ;if r == 1, the s ← 1
```


Example: Hexadecimal Character Conversion

- We would like to convert 4 binary bits to hexadecimal digits
- Assume that these 4 bits are stored at LSB of `r0` and the rest of bits are zeros
- Note that the ASCII code of
  - '0' is 48, i.e., 0x30 (difference from 0000_2 is = 0x30)
  - '1' is 49, i.e., 0x31 (difference from 0001_2 is = 0x30)
  - ...
  - '9' is 57, i.e., 0x39 (difference from 1001_2 is = 0x30)
- Note that the ASCII code of
  - 'A' is 65, i.e., 0x41 (difference from 1010_2 is = 0x37)
  - 'B' is 66, i.e., 0x42 (difference from 1011_2 is = 0x37)
  - ...
  - 'F' is 70, i.e., 0x46 (difference from 1111_2 is = 0x37)
- The conversion algorithm is:
- `character = hexValue + 0x30`

```
if(character > 0x39)
  character += 7
```

ADDGT not ADDGE
Not correct in the book page 202

```
ADD   r0,r0,#0x30;add 0x30 to convert 0 through 9 to ASCII
CMP   r0,#0x39   ;check for A to F hex values
ADDGT r0,r0,#7   ;If A to F, then add 7 to get the ASCII
```

0000 →	0
0001 →	1
0010 →	2
0011 →	3
0100 →	4
0101 →	5
0110 →	6
0111 →	7
1000 →	8
1001 →	9
1010 →	A
1011 →	B
1100 →	C
1101 →	D
1110 →	E
1111 →	F

Example: Hexadecimal Character Conversion

- We would like to convert 4 binary bits to hexadecimal digits
- Assume that these 4 bits are stored at LSB of `r0` and the rest of bits are zeros
- Note that the ASCII code of
  - `'0'` is 48, i.e., `0x30` (difference from 0000_2 is = `0x30`)
  - `'1'` is 49, i.e., `0x31` (difference from 0001_2 is = `0x30`)
  - ...
  - `'9'` is 57, i.e., `0x39` (difference from 1001_2 is = `0x30`)
- Note that the ASCII code of
  - `'A'` is 65, i.e., `0x41` (difference from 1010_2 is = `0x37`)
  - `'B'` is 66, i.e., `0x42` (difference from 1011_2 is = `0x37`)
  - ...
  - `'F'` is 70, i.e., `0x46` (difference from 1111_2 is = `0x37`)
- Another algorithm:
- `character = hexValue +(hexValue <= 0x9)? 0x30 : 0x37;`

```
CMP    r0,#0x9    ;is it 0-9 or A-F hex values?
ADDLE r0,r0,#0x30;if it is 0-9, add 0x30 to convert to ASCII
ADDGT r0,r0,#0x37;if it is A-F, add 0x37 to convert to ASCII
```

```
0000 → 0
0001 → 1
0010 → 2
0011 → 3
0100 → 4
0101 → 5
0110 → 6
0111 → 7
1000 → 8
1001 → 9
1010 → A
1011 → B
1100 → C
1101 → D
1110 → E
1111 → F
```


Example: Multiple Selection

- Let us translate the following C code

```
switch (i)
{ case  0: do action; break;
  case  1: do action; break;
  ...
  case  N: do action; break;
  default: do something;
}
```

- Assume that `r0` contains the selector `i`

```
    ADR r1, TBL          ;r1 ← the address of the jump table
    CMP r0, N            ;is the switch variable in range?
    ADDLE pc,r1,r0,LSL#2 ;If OK, jump to the appropriate case
    ;The default action goes here

    ...
TBL B case0
    B case1
    ...
    B caseN
```

Example: Multiple Selection

- Let us translate the following C code

```
switch (i)
{ case  0: do action; break;
  case  1: do action; break;
  ...
  case  N: do action; break;
  default: do something;
}
```

- Assume that `r0` contains the selector `i`

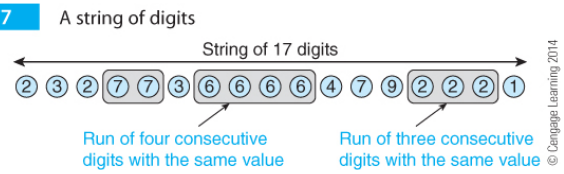
```
           TEQ r0, 0 ;is the switch variable == case 0?
           BEQ case0 ;If case 0, jump to the case0 code
           TEQ r0, 1 ;is the switch variable == case 1?
           BEQ case1 ;If case 1, jump to the case0 code
           ...
           TEQ r0, N ;is the switch variable == case N?
           BEQ caseN ;If case N, jump to the caseN code
           B default
Case0      do action of case 0
           B AfterCase
Case1      do action of case 1
           B AfterCase
           ...
CaseN      do action of case N
           B AfterCase
default    do action of default
AfterCase  ...
```

The order of cases/default depends on the actual order in your program.

Example: Finding the Longest Sequence of Repeated Digits

- In Chapter one, we attempted to find the longest sequence of repeated digits.

FIGURE 1.7 A string of digits



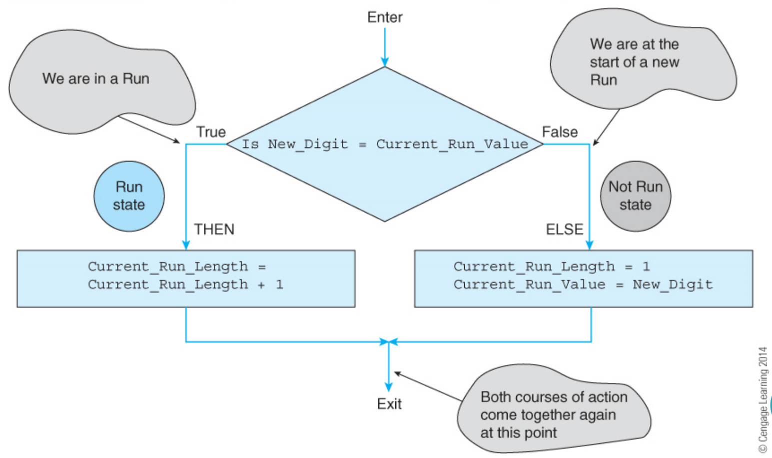
- Let us revisit this problem and implement the solution using ARM assembly language.
- If you recall, we proposed 13 steps to solve this problem:

1. Read the first digit in the string and call it New_Digit
2. Set the Current_Run_Value to New_Digit
3. Set the Current_Run_Length to 1
4. Set the Max_Run to 1
5. REPEAT
6.   Read the next digit in the sequence (i.e., read a New_Digit)
7.   IF its value is the same as Current_Run_Value
8.   THEN Current_Run_Length = Current_Run_Length + 1
9.   ELSE {Current_Run_Length = 1
10.   Current_Run_Value = New_Digit}
11.   IF Current_Run_Length > Max_Run
12.   THEN Max_Run = Current_Run_Length
13. UNTIL The last digit is read

Example: Finding the Longest Sequence of Repeated Digits

- Inside the body of the loop, there is an IF…THEN…ELSE construct that we used to test whether we are in a run or not to either increment the run length or reset it to 1

FIGURE 1.10 Illustrating the IF...THEN...ELSE construct graphically

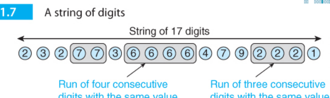


213

Computer Organization and Architecture: Themes and Variations, 1st Edition Clements

Example: Finding the Longest Sequence of Repeated Digits

FIGURE 1.7 A string of digits




```
        AREA   RunLength,CODE,READONLY
        ENTRY
        ADR    r9, String ;r9 points to the sting
        LDRB   r0, EoS    ;r0 is the EoS symbol
        LDRB   r1,[r9],#1 ;Step-01: r1 is New_Dig
        MOV    r2,r1      ;Step-02: r2 is the Current_Run_Value
        MOV    r3,#1      ;Step-03: r3 is the Current_Run_Length (set to 1)
        MOV    r4,#1      ;Step-04: r4 is the Max_Run_Length (set to 1)
Repeat  LDRB   r1,[r9],#1 ;Step-05 & 06: REPEAT: Read next digit
        CMP    r2,r1      ;Step-07:  Compare New_Digit and Current_Digit
        ADDEQ  r3,r3,#1   ;Step-08:  IF same THEN Current_Length=Current_Length+1
        MOVNE  r3,#1      ;Step-09:          ELSE Current_Run_Length = 1
        MOVNE  r2,r1      ;Step-10:               Current_Run_Value  = New_Digit
        CMP    r3,r4      ;Step-11:  IF Current_Run_Length > Max_Run
        MOVPL  r4,r3      ;Step-12:  THEN Max_Run = Current_Run_Length
        TEQ    r0,r1      ;Step-13:  Testing the end of string
        BNE Repeat        ;Step-13: UNTIL all digits tested
Park    B Park            ;parking loop
String  DCB 2,3,2,7,7
        DCB 3,6,6,6,6,4
        DCB,7,9,2,2,2,1
EoS     DCB 0xFF
        END
```

1. Read the first digit in the string and call it New_Digit
2. Set the Current_Run_Value to New_Digit
3. Set the Current_Run_Length to 1
4. Set the Max_Run to 1
5. REPEAT
6. Read the next digit in the sequence (i.e., read a New_Digit)
7. IF its value is the same as Current_Run_Value
8. THEN Current_Run_Length = Current_Run_Length + 1
9. ELSE {Current_Run_Length = 1
10. Current_Run_Value = New_Digit}
11. IF Current_Run_Length > Max_Run
12. THEN Max_Run = Current_Run_Length
13. UNTIL The last digit is read

214