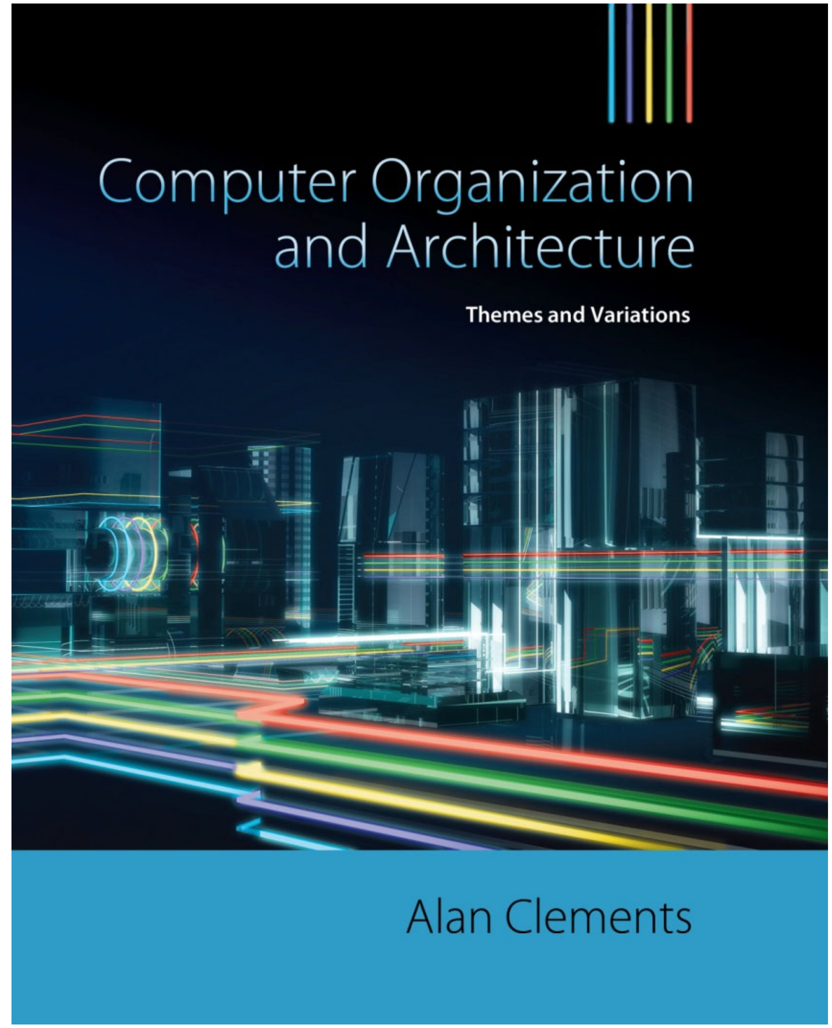
Part A

CHAPTER 3

Architecture and Organization



These slides are being provided with permission from the copyright for CS2208 use only. The slides must not be reproduced or provided to anyone outside of the class.

All download copies of the slides are for personal use only.
Students must destroy these copies within 30 days after receipt of final course evaluations.

CENGAGE Learning

Example 1: 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`?
  CMP updates all flags while TEQ doesn't.

To know the difference, read slide #72

- What is the difference between `RSBMI` and `RSBLT`?
  MI: N flag only LT: N clear V set / N set V clear.

To know the difference, read slide #83

- Can we use `RSBMI r0, #0` instead of `RSBMI r0, r0, #0` ?
  Yes. It is the same thing.

To know the answer, read slide #59

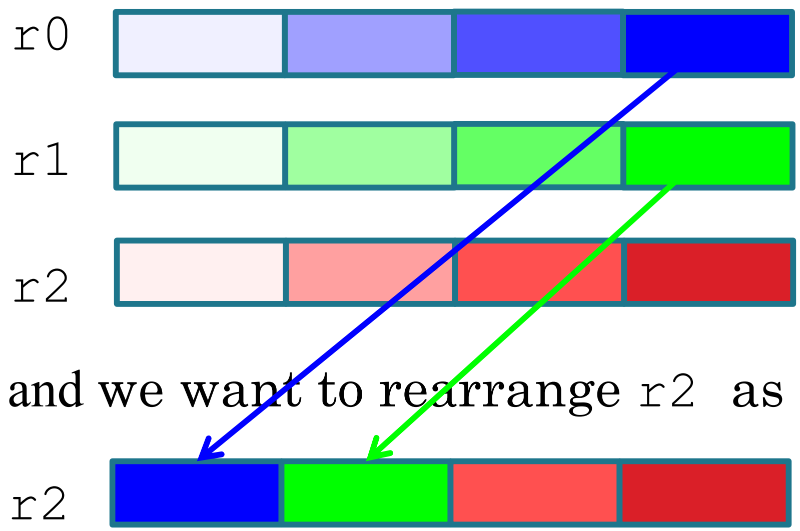
- Can we use `NEGMI r0, r0` instead of `RSBMI r0, r0, #0` ?
  Yes.
  pseudo instruction
  NEGMI r0, r0
  is implemented as
  RSBMI r0, r0, #0

To know the answer, read slide #59

Example 2: Byte Manipulation and Concatenation

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

BIC cannot use for PC.



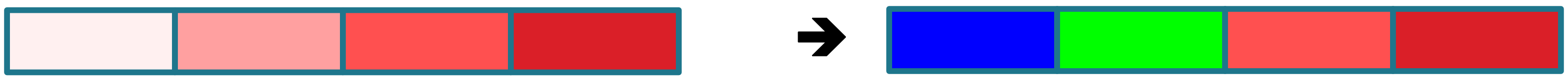
and we want to rearrange r2 as follow:

Note that: we can not do:
BIC **r2**,r2,#0xFFFF0000
To know the reason, read
Slides 105-110

```
AND r0,r0,#0xFF            ;clear all high order 3 bytes
AND r1,r1,#0xFF            ;clear all high order 3 bytes
BIC r2,r2,#0xFF0000        ;clear the 3rd byte
BIC r2,r2,#0xFF000000      ;clear the 4th byte
ADD r2,r2,r1,LSL#16        ;LSL r1 by 2 bytes & add it to r2
ADD r2,r2,r0,LSL#24        ;LSL r0 by 3 bytes & add it to r2
```

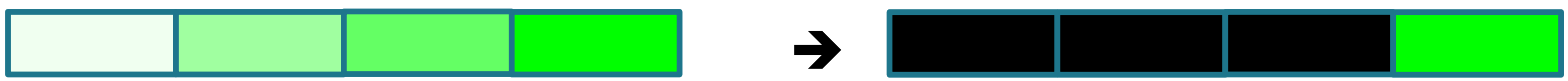
AND r2, r2, #0x0000FFFF ?

Example 2: Byte Manipulation and Concatenation

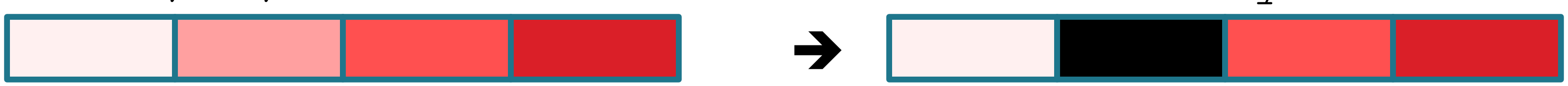


```
AND r0,r0,#0xFF          ;clear all high order 3 bytes
```

```
AND r1,r1,#0xFF          ;clear all high order 3 bytes
```

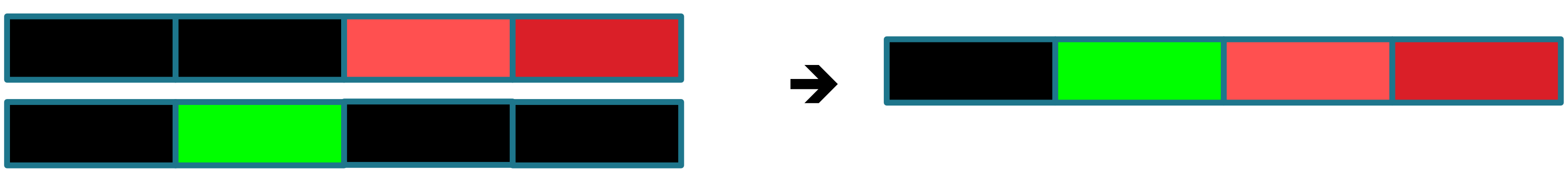


```
BIC r2,r2,#0xFF0000      ;clear the 3rd byte
```

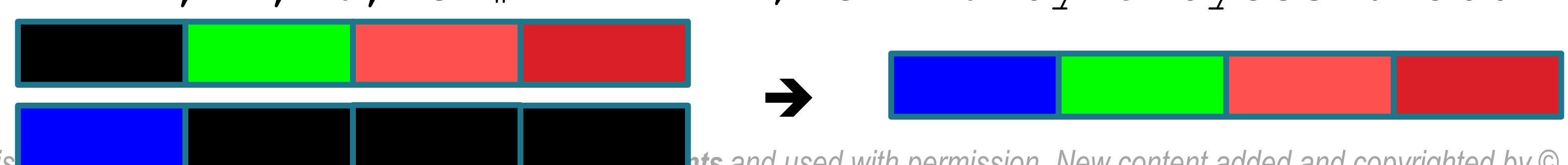


```
BIC r2,r2,#0xFF000000    ;clear the 4th byte
```

```
ADD r2,r2,r1,LSL#16 => 2×8   ;LSL r1 by 2 bytes & add it to r2
```

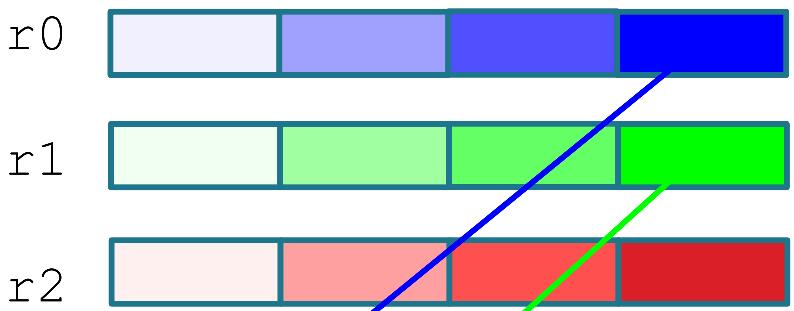


```
ADD r2,r2,r0,LSL#24      ;LSL r0 by 3 bytes & add it to r2
```



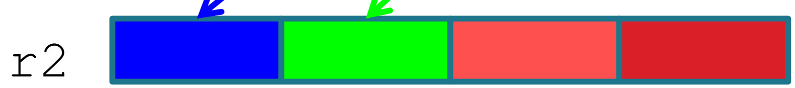
Example 2: Byte Manipulation and Concatenation

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



AND R0, 0xFF
AND R1, 0xFF
BIC R2, 0xFF0000
BIC R2, 0xFF000000
AND R2, R2, R0, LSL#24
AND R2, R2, R1, LSL#16.

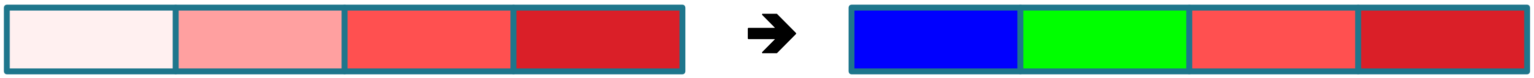
and we want to rearrange `r2` as follow:



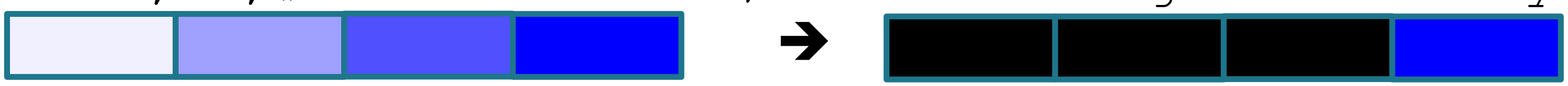
- ***Another solution in 5 instructions***

```
AND r0,r0,#0xFF       ;clear r0 all high order 3 bytes
AND r1,r1,#0xFF       ;clear r1 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
```

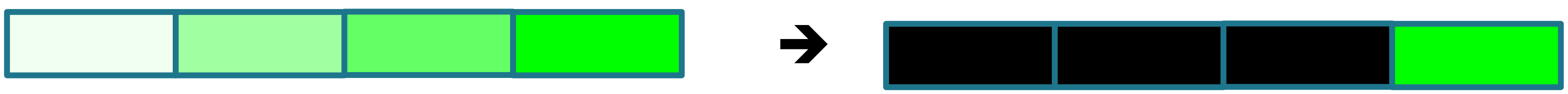
Example 2: Byte Manipulation and Concatenation



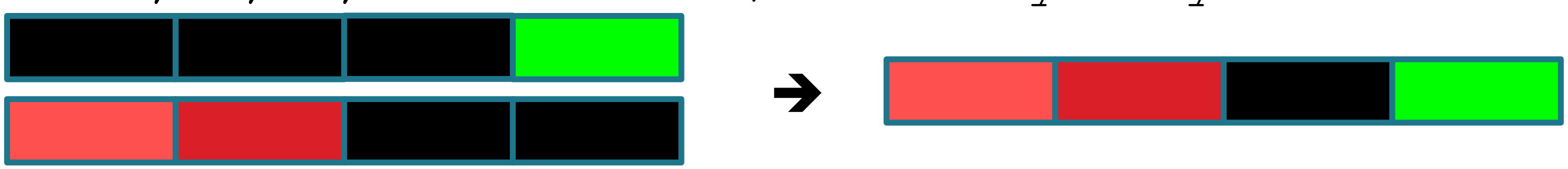
```
AND r0,r0,#0xFF          ;clear all high order 3 bytes
```



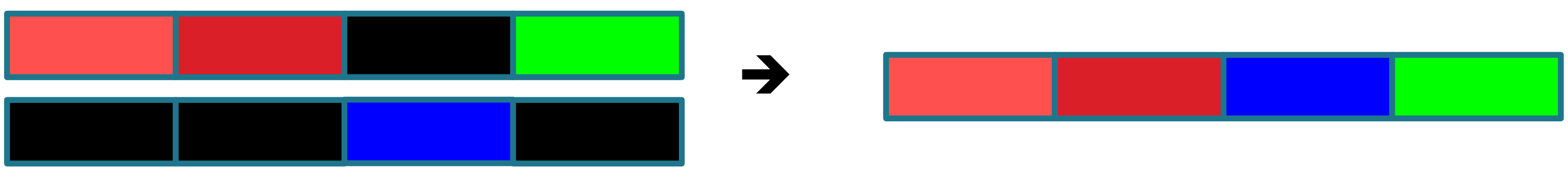
```
AND r1,r1,#0xFF          ;clear all high order 3 bytes
```



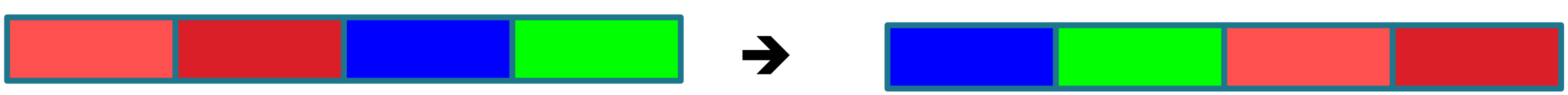
```
ADD r2,r1,r2,LSL#16      ;LSL r2 by 2 bytes & add r1 to it
```



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

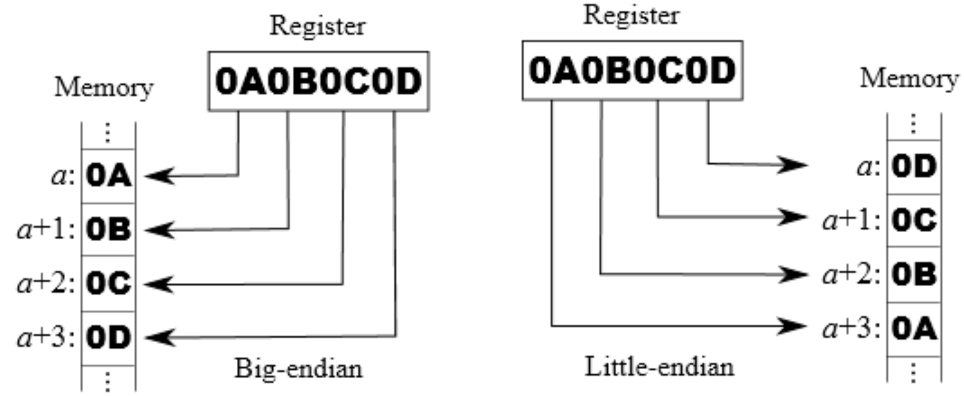


```
MOV r2,r2,ROR#16         ;Swap the two r2 16 bits together
```



Example 3: Byte Reversal (Big-endian ⇔ Little-endian)

- Suppose that **0xAB CD EF GH** is stored in `r0`
- We want to reverse the content of `r0`, `i.e.,` store **0xGH EF CD AB** in `r0`



- Let us review the XOR truth table
  - x ⊕ x = 0
  - x ⊕ 0 = x
  - x ⊕ y ⊕ y = x

- We will use `r1` as a working register

0xEFGHABCD

```
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⊕0,H⊕0,A⊕A⊕E,B⊕B⊕F,C⊕0,D⊕0,E⊕E⊕A,F⊕F⊕B
                        ; G , H ,E    ,F    ,C  ,D  ,A    ,B
```

A	B	C = A ⊕ B
0	0	0
0	1	1
1	0	1
1	1	0

Example 4: 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**

The red values are the originals.

```
ADD r0,r0,r1    ; [r0] ← [r0] + [r1]
SUB r1,r0,r1    ; [r1] ← [r0] - [r1]
                ; [r1] ←([r0] + [r1]) - [r1]
                ; [r1] ← [r0]
SUB r0,r0,r1    ; [r0] ← [r0] - [r1]
                ; [r0] ←([r0] + [r1]) - [r0]
                ; [r0] ← [r1]
```

```
X ← X + Y
Y ← X - Y
X ← X - Y
```

$x' = x + y$

$y' = x' - y = x + y - y = x$

$x'' = x' - y' = x + y - x = y.$

shift done

Example 4: 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**
- ***Another solution***

  Let us review the XOR truth table

  - x ⊕ x = 0
  - x ⊕ 0 = x
  - **x ⊕ y ⊕ y = x**

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

The red values are the originals.

```
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 5: 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:

Write one ARM instruction to store `r1 × 16` into `r2`

```
MOV r2, r1, LSL#4       ; [r2]←[r1] × 2^4
```

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

Write one ARM instruction to store `r1 × 17` into `r2`

```
ADD r2, r1, r1, LSL#4 ; [r2]←[r1] + [r1] × 2^4
```

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

Write one ARM instruction to store `r1 × 15` into `r2`

```
RSB r2, r1, r1, LSL#4 ; [r2]←[r1] × 2^4  - [r1]
```

Example 5: 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
```

condition

operation.

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

Example 6: Dividing by D

- Dividing by **D** can be implemented using MUL and ASR instructions

- Example:
  Write ARM instructions to divide **r0** by **D** and store the result in **r1**
  i.e., **[r1] ← [r0] / D**

- The result can be written as:

```
[r0] / D      = [r0] × (1 / D)
              = [r0] × (2^N/D) / 2^N
```

  - ✓ Select **N** to be a large integer at the same time not to cause an overflow when evaluating **[r0] × (2^N/D)**
  - ✓ Evaluate **[r0] × (2^N/D)**
  - ✓ Arithmetic shift right the result N time

- If **D = 5** and **r0 = 32004**, we can pick **N = 16**
- **2^N / D** = **2^16 / 5** = **1024 × 64 / 5** = **13107.2**
  round(**13107.2**) = **13107**

Note that 13107 / 2^16 = 0.199997 ≈ 0.2

```
LDR r2,=13107;  (2^N/D)
MUL r1,r2,r0 ;  [r0] × (2^N/D)
ASR r1,#16   ;  [r0] × (2^N/D) / 2^N = [r0] / D
```

Example 7: 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'               ;Is it in the range of the capital?
RSBGES r1, r0, #'Z'            ;If >= 'A',
                               ;then check with 'Z'
                               ;     and update the flags
ORRGE  r0, r0, #2_100000       ;If between 'A' and 'Z' inclusive,
                               ;then set bit 5 to force lower case
```

Example 8: If Statement in One Instruction!!

- Let us translate the following C code

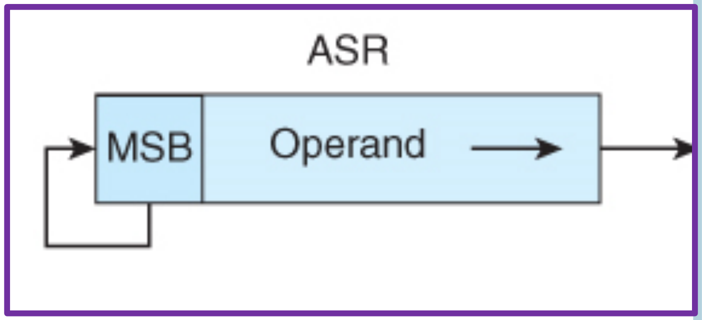
```
if(x < 0)
  x = 0;
```

- 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 9: 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;
```

```
TST    r0,#0x8      ;check the value of bit p
TSTNE r0,#0x2      ;if p == 1,
                    ;   check the value of bit r
ORRNE  r0,r0,#1    ;if r == 1,
                    ;   set s ← 1
```

Example 10: Hexadecimal Character Conversion

- We would like to convert **4 binary bits** to **hexadecimal digits**
- Assume that these 4 bits are stored at the LSBs of `r0` and the rest of the 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 also 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 = the4BitBinaryValue + 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 10: Hexadecimal Character Conversion

- We would like to convert **4 binary bits** to **hexadecimal digits**
- Assume that these 4 bits are stored at the LSB of `r0` and the rest of the 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 also 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`)

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'

- Another algorithm:

```
character = the4BitBinaryValue
    +(the4BitBinaryValue <= 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
```

Example 11: 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;
}
```

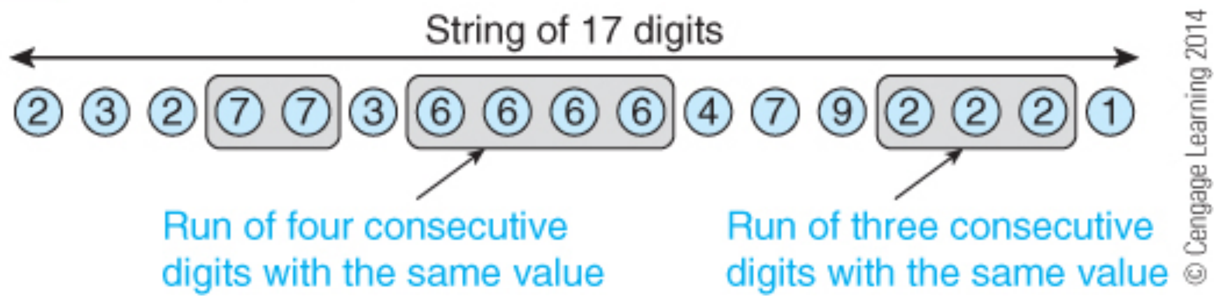
- Assume that `r0` contains the selector `i`

```
          TEQ r0, 0 ;is the switch variable == 0?
          BEQ case0 ;If i == 0, jump to the case0 code
          TEQ r0, 1 ;is the switch variable == 1?
          BEQ case1 ;If i == 1, jump to the case1 code
          ...
          TEQ r0, N ;is the switch variable == N?
          BEQ caseN ;If i == 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 ...
```

Example 12: 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



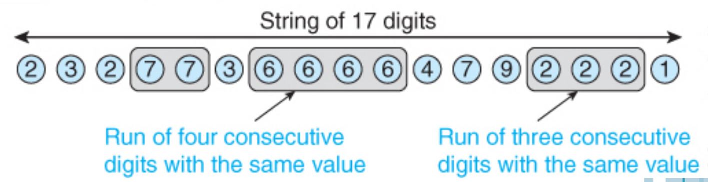
- Let us revisit this problem and implement the solution using ARM assembly language.
- If you recall, we proposed 13 steps to solve this problem:

```
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 12: Finding the Longest Sequence of Repeated 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_Digit
       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 (i.e., New_Digit)
       CMP    r1,r2      ;Step-07:  Compare New_Digit and Current_Run_Value
       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
       MOVGT  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
```

FIGURE 1.7 A string of digits




```
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
```