Applications of Bit Manipulation

Leo Villareal's 23,000 Points of Light Illuminate The Renwick Gallery, Washington D.C.

Villareal and his team have developed custom tools and software over the past decade, which ensures that the sequences never repeat, keeping things from getting predictable, even for the artist himself. [1]

MULTIPLICATION IN ASSEMBLY LANGUAGE

- 4 bit multiplication
  - Shift the multiplier to the right.
  - If CF=1
    - add the multiplicand to the result.
  - Shift the multiplicand to the left.
  - Repeat the algorithm 4 times.
- For 4 bit multiplication, multiplicand and result should be 8 bit.
- Similarly, for n bit multiplication, multiplicand and result should be 2*n bits.

```
   1101 = 13        Accumulated Result
   0101 = 5
   -----                           0 (Initial Value)
   1101 = 13        0 + 13  = 13
  0000x = 0         13 + 0  = 13
 1101xx = 52        13 + 52 = 65
0000xxx = 0         65 + 0  = 65 (Answer)
```

```
                 01111011      123
    ×            00100100      36
    ---------------------
               01111011        123 SHL 2
+           01111011           123 SHL 5
    ---------------------
     0001000101001100          4428
```

Example 4.1

```
; 4bit multiplication algorithm
[org 0x100]
              jmp  start

multiplicand: db   13                  ; 4bit multiplicand (8bit space)
multiplier:   db   5                   ; 4bit multiplier
result:       db   0                   ; 8bit result

start:        mov  cl, 4               ; initialize bit count to four
              mov  bl, [multiplicand]  ; load multiplicand in bl
              mov  dl, [multiplier]    ; load multiplier in dl

checkbit:     shr  dl, 1               ; move right most bit in carry
              jnc  skip                ; skip addition if bit is zero

              add  [result], bl        ; accumulate result

skip:         shl  bl, 1               ; shift multiplicand left
              dec  cl                  ; decrement bit count
              jnz  checkbit            ; repeat if bits left

              mov  ax, 0x4c00          ; terminate program
              int  0x21
```

Question

- Modify the algorithm given in last slide for 8 bit multiplication.
- How will you modify the algorithm given in last slide for 16 or 32 bit multiplication?

EXTENDED OPERATIONS

- Working with larger numbers
- For example
  - Adding/Subtracting 32 bit or 64 bit numbers
  - Multiplying 16 or 32 or 64 bit numbers
  - Shifting 32 bit number

Extended Multiplication

- Take an example of multiplying two 16 bit numbers.
- The result will need 32 bits and multiplicand will also need 32 bits.
- This will require SHL 32 bit multiplicand and addition of 32 bit multiplicand and 32 bit result.
- So to perform extended multiplication we should first look at extended addition and extended shifting.
    - *Reminder the one instruction of add or SHR only works for 8 or 16 bits*

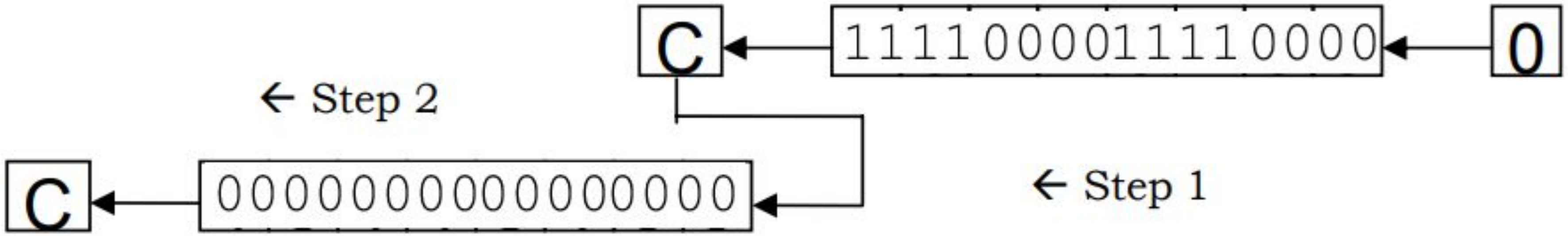
Extended Shifting Left

- Consider a 32 it number num1
- `Num1: dd  0000 0000 0000 0000 1111 0000 1111 0000b`
  - Note spaces are just there for readability.
- How will you Shift it left by 1 such that the result is
- `Num1: dd  0000 0000 0000 0001 1110 0001 1110 0000b`

Extended Shifting Left

- You will use two instructions
  - `Shl word [num1], 1`
  - `Rcl word [num1+2], 1`
- Effect of 1st instruction: `Shl word [num1], 1`
- `Num1: dd  0000 0000 0000 0000` **1110 0001 1110 0000** `b`
- `CF=1`
- Effect of 2nd Instruction `Rcl word [num1+2], 1`
- `Num1: dd` **0000 0000 0000 0001** `1110 0001 1110 0000 b`
- `CF= 0`

Extended Shifting left



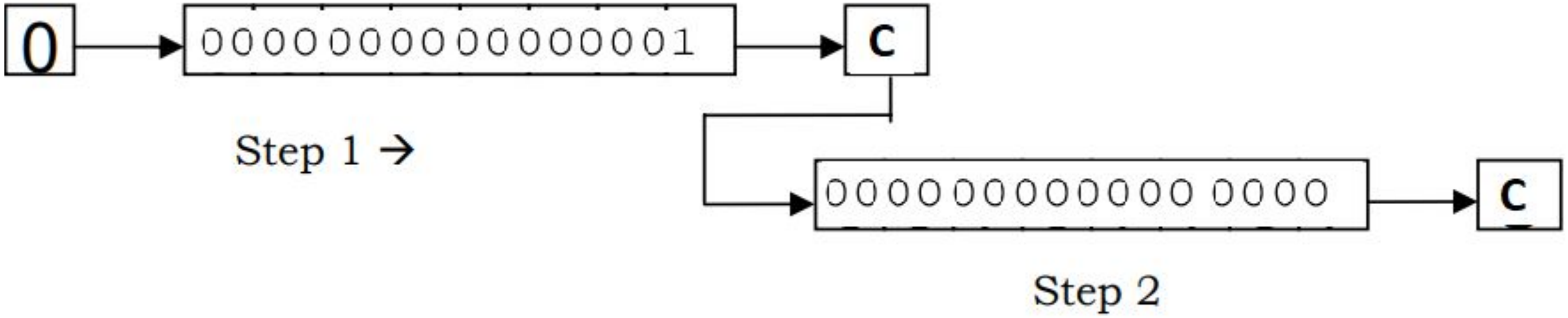
Extended Shifting Right

- Consider a 32 it number num1
- `Num1: dd  0000 0000 0000 0001 0000 0000 0000 0000b`
  - Note spaces are just there for readability.
- How will you Shift it Right by 1 such that the result is
- `Num1: dd  0000 0000 0000 0000 1000 000 0000 0000b`

Extended Shifting Right

- You will use two instructions
  - `Shr word [num1+2], 1`
  - `Rcr word [num1], 1`
- Effect of 1st instruction: `Shr word [num1+2], 1`
- Num1: dd **0000 0000 0000 0000** 0000 0000 0000 0000b
- CF=1
- Effect of 2nd Instruction `Rcr word [num1], 1`
- Num1: dd 0000 0000 0000 0000 **1000 0000 0000 0000** b
- CF= 0

Extended Shifting Right

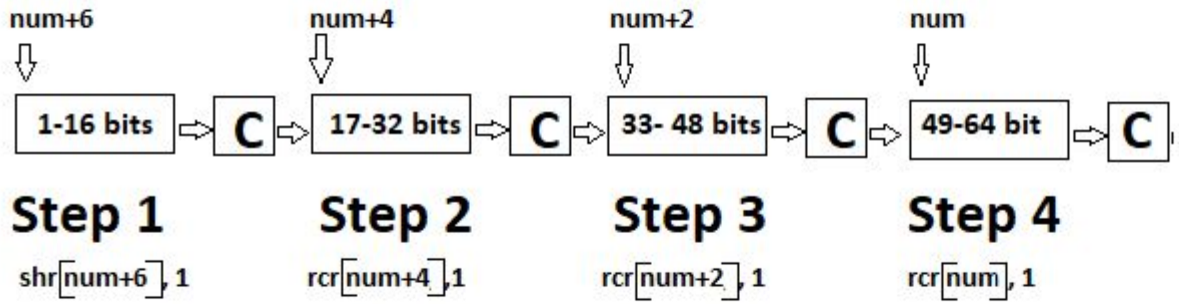


Extended Shift

- How will you shift right a 64 bit number?

Extended Shift

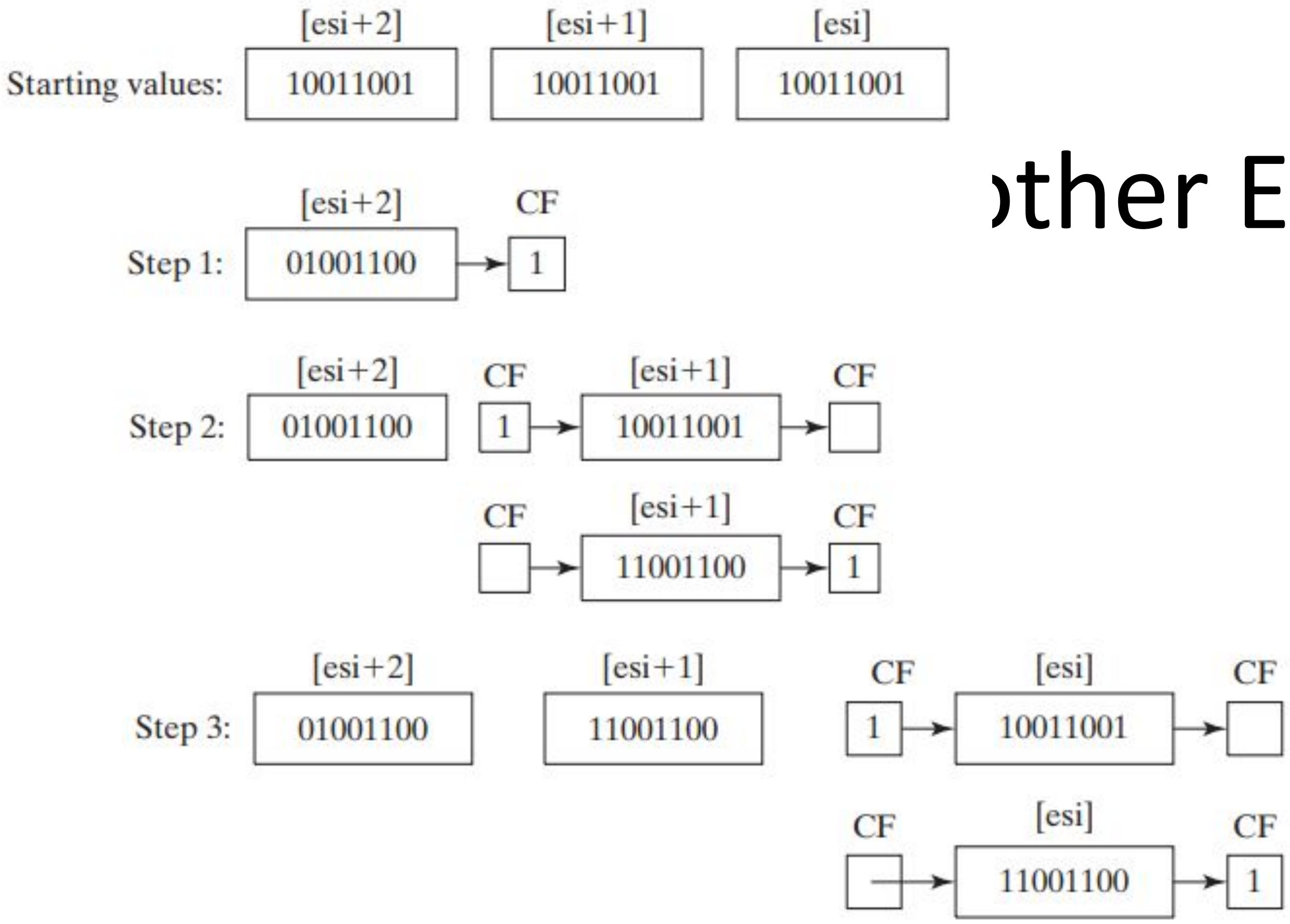
- How will you shift right a 64 bit number?
- Answer



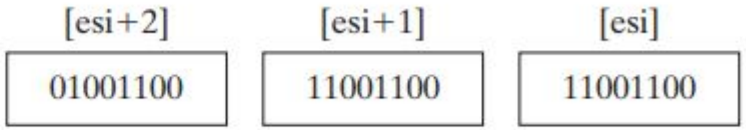
Extended Shift

- How will you shift left a 64 bit number?

other Example



After Step 3 is complete, all bits have been shifted 1 position to the right:



Extended Addition

- ADC, Addition with Carry will help use to perform extended addition.
- How ADC works
  - ADC ax, bx; this is equivalent to ax+bx+CF

```
[org 0x0100]

    stc
    mov al, 3
    adc al, 0 ; al=4

    clc
    mov al, 3
    adc al, 0 ; al=3

    mov ax, 0x4c00
    int 0x21
```

Extended Addition

- For adding numbers with n words
- Add nth Words of first and second operand
- For i in n-1 to 1
    - add with carry the ith word of first and ith word of second operand

Extended Addition (Example)

- Example:
- Adding 64 bits

Initially				
num1:	1000 1000 0000 0000	0110 0000 1111 1111	0100 0000 0000 0000	1111 1111 1111 1111
num2:	1000 1111 0000 1111	0000 0000 0000 0000	0100 0000 0000 0001	1000 0000 0000 0000
result:	0000 0000 0000 0000	0000 0000 0000 0000	0000 0000 0000 0000	0000 0000 0000 0000

Extended Addition (Example)

- Step 1
  - Mov ax, [num1]
  - mov bx, [num2]
  - Add ax, bx
  - Mov [result], ax

After Step 1: CF=1				
num1:	1000 1000 0000 0000	1110 0000 1111 1111	0100 0000 0000 0000	1111 1111 1111 1111
num2:	1000 1111 0000 1111	1000 0000 0000 0000	0100 0000 0000 0001	1000 0000 0000 0000
result:	0000 0000 0000 0000	0000 0000 0000 0000	0000 0000 0000 0000	0111 1111 1111 1111

Extended Addition (Example)

- Step 2
  - Mov ax, [num1+2]
  - mov bx, [num2+2]
  - ADC ax, bx
  - Mov [result+2], ax

After Step 1: CF=0				
num1:	1000 1000 0000 0000	1110 0000 1111 1111	0100 0000 0000 0000	1111 1111 1111 1111
num2:	1000 1111 0000 1111	1000 0000 0000 0000	0100 0000 0000 0001	1000 0000 0000 0000
result:	0000 0000 0000 0000	0000 0000 0000 0000	1000 0000 0000 0010	0111 1111 1111 1111

Extended Addition (Example)

- Step 3
  - Mov ax, [num1+4]
  - mov bx, [num2+4]
  - ADC ax, bx
  - Mov [result+4], ax

After Step 1: CF=1				
num1:	1000 1000 0000 0000	1110 0000 1111 1111	0100 0000 0000 0000	1111 1111 1111 1111
num2:	1000 1111 0000 1111	1000 0000 0000 0000	0100 0000 0000 0001	1000 0000 0000 0000
result:	0000 0000 0000 0000	0110 0000 1111 1111	1000 0000 0000 0010	0111 1111 1111 1111

Extended Addition (Example)

- Step 4
  - Mov ax, [num1+6]
  - mov bx, [num2+6]
  - ADC ax, bx
  - Mov [result+6], ax

After Step 4: CF=1				
num1:	1000 1000 0000 0000	1110 0000 1111 1111	0100 0000 0000 0000	1111 1111 1111 1111
num2:	1000 1111 0000 1111	1000 0000 0000 0000	0100 0000 0000 0001	1000 0000 0000 0000
result:	0001 0111 0001 0000	0110 0000 1111 1111	1000 0000 0000 0010	0111 1111 1111 1111

- So the result of addition is
- CF=1 , 0001 0111 0001 0000 0110 0000 1111 1111 1000 0000 0000 0010 0111 1111 1111 1111
- Note that the carry of the Most significant word is stored in CF using these 4 steps

Extended Subtraction

- For subtraction the same logic will be used and just like addition with carry there is an instruction to subtract with borrows called SBB.
- SBB ax, bx; this is equivalent to ax-bx-CF

```
stc
mov al, 3
sbb al, 0 ; al=2

clc
mov al, 3
sbb al, 0 ; al=3
```

Extended Subtraction

- For subtracting numbers with n words
- Sub nth Words of first and second operand
- For i in n-1 to 1
    - Subtract with borrow the ith word of first and ith word of second operand

Extended Subtraction (Exercise)

- Example:
- Subtracting 64 bits

Initially				
num1:	1000 1000 0000 0000	0110 0000 1111 1111	0100 0000 0000 0000	1111 1111 1111 1111
num2:	1000 1111 0000 1111	0000 0000 0000 0000	0100 0000 0000 0001	1000 0000 0000 0000
result:	0000 0000 0000 0000	0000 0000 0000 0000	0000 0000 0000 0000	0000 0000 0000 0000

- Work out the given subtraction as we did addition, show each step

Question

- Give two operands of n words, if you know the value of n write the code of adding these two operands in loop
- Give two operands of n words, if you know the value of n write the code of subtracting these two operands in loop

Extended Multiplication

- On slide 4 we saw an example to multiple 4 bit numbers.
- The same algorithm can be now used to multiply a numbers of any size.
- The algorithm was as follow
  - Shift the multiplier to the right. (now you know how to use extended shift)
  - If CF=1
    - add the multiplicand to the result. (now you know how to use extended addition)
  - Shift the multiplicand to the left.
  - Repeat the algorithm n times. (where n is size of multiplier)

Example

Example 4.2

```
; 16bit multiplication
[org 0x0100]
              jmp  start

multiplicand: dd   1300              ; 16bit multiplicand 32bit space
multiplier:   dw   500               ; 16bit multiplier
result:       dd   0                 ; 32bit result

start:        mov  cl, 16            ; initialize bit count to 16
              mov  dx, [multiplier]  ; load multiplier in dx

checkbit:     shr  dx, 1             ; move right most bit in carry
              jnc  skip              ; skip addition if bit is zero

              mov  ax, [multiplicand]
              add  [result], ax      ; add less significant word
              mov  ax, [multiplicand+2]
              adc  [result+2], ax    ; add more significant word

skip:         shl  word [multiplicand], 1
              rcl  word [multiplicand+2], 1 ; shift multiplicand left
              dec  cl                ; decrement bit count
              jnz  checkbit          ; repeat if bits left

              mov  ax, 0x4c00        ; terminate program
              int  0x21
```

Example

- Change the code give in previous slide to work for 32 bit multiplication i.e. the result should be 64 bit

BITWISE LOGICAL OPERATIONS

BITWISE LOGICAL OPERATIONS

- AND operation
  - Examples are “and ax, bx” and “and byte [mem], 5.”
  - All possibilities that are legal for addition are also legal for the AND operation. The different thing is the bitwise behavior of this operation.
- OR operation
  - Examples are “or ax, bx” and “or byte [mem], 5.”
- XOR operation
  - Examples are “xor ax, bx” and “xor byte [mem], 5
- NOT operation
  - Examples are “not ax” and “not byte [mem]”.

MASKING OPERATIONS

- Selective Bit Clearing
  - Done using AND operations
  - Example to clear the LSB in AL, used AND AX, 11111110b
  - This operation is called masking, 11111110 was mask in this example
- Selective Bit Setting
  - Done using OR operations
  - Example to set the LSB in AL, used AND AX, 00000001b
  - 00000001b is the mask here
- Selective Bit Inversion
  - Done using XOR
  - For example to toggle LSB and MSB in AL use XOR AL, 1000 0001b

MASKING OPERATION

Selective Bit Testing

- AND operation can be used to test if a certain bit in a number of on
  - But this will change the mask or the operand
- TEST instruction is a non destructive alternative for selective bit testing.
- It doesn’t change the destination and only sets the sign, Zero and parity and flags as would have AND operation

```
mov al, 00010001b
test al,00001001b; ZF=0
test al,00010000b; ZF=0
test al,00000010b; ZF=1
```

- Next slide shows the use of test in multiplication algorithms so that multiplier is retained

Example 4.3

```
; 16bit multiplication using test for bit testing
[org 0x0100]
              jmp  start

multiplicand: dd   1300                ; 16bit multiplicand 32bit space
multiplier:   dw   500                 ; 16bit multiplier
result:       dd   0                   ; 32bit result

start:        mov  cl, 16              ; initialize bit count to 16
              mov  bx, 1               ; initialize bit mask

checkbit:     test bx, [multiplier]    ; test right most bit
              jz   skip                ; skip addition if bit is zero

              mov  ax, [multiplicand]
              add  [result], ax        ; add less significant word
              mov  ax, [multiplicand+2]
              adc  [result+2], ax      ; add more significant word

skip:         shl  word [multiplicand], 1
              rcl  word [multiplicand+2], 1 ; shift multiplicand left
              shl  bx, 1               ; shift mask towards next bit
              dec  cl                  ; decrement bit count
              jnz  checkbit            ; repeat if bits left

              mov  ax, 0x4c00          ; terminate program
              int  0x21
```

Reading

- Chapter 4 Bilal Hashmi