CYBV 471 Assembly Programming for Security Professionals
Week 7

Math in Assembly Language

Agenda

- **Flag Register**
- **Math in Assembly Language**
  - Addition instruction
  - Substation instruction
  - Increment instruction
  - Decrement instruction
  - NOT instruction
  - NEG instruction
  - Unsigned Integer Division
  - Signed Integer Division
  - Unsigned Integer Multiplication
  - Signed Integer Multiplication

- Unsigned Integers: Positive or zero
- n-bits can have a value between 0 and $(2^n - 1)$, (2^n different values)
- For 8 bits, Max. positive value is $(2^8 - 1) = 255$
- Signed Integers:
  - Positive, zero, or negative values.
  - Negative values are stored as 2's complement (MSB is sign value)

Positive Value

0	0	0	0	0	0	0	0

Sign bit

Min. value =0

0	1	1	1	1	1	1	1

Max. Positive Value $127 = (2^7 - 1) = (2^{(n-1)} - 1)$

Negative Value

1	1	1	1	1	1	1	1

1	0	0	0	0	0	0	0

1s complment

0	0	0	0	0	0	0	0

0	1	1	1	1	1	1	1

2's complment (+1)

0	0	0	0	0	0	0	1

(-1)

$+ 1$

1	0	0	0	0	0	0	0

(-128)

Signed numbers values:

$-2^{(n-1)}$ to $(2^{(n-1)} - 1)$

$-128 = -2^7$

For n bits. Min. nevative value $= -2^{(n-1)}$

- Signed short integer (16 bits) can hold decimal values from -32,768 to +32,767 (2^{15})
- Signed long integer (32 bits) can contain values from

  -2,147,483,648 to +2,147,483,647
- Signed double integer (64 bits) can represent decimal values from

  -9,223,372,036,854,775,808 to +9,223,372,036,854,775,807

EFLAG-Flag Register

- The flag register (EFLAG) is a status register
- It is 32 bits in size, each bit is a flag
- Each flag (bit) is **set** (=1) or **cleared** (= 0)
  - To control CPU operations or
  - To indicate the results of CPU operations
- Examples
- **ZF** (Zero Flag)
  Set (=1) when the result of an operation is zero
  Reset (=0) when the result of an operation is not zero

- **SF** (Sign Flag)
  - If (SF =1), indicates the result is negative
  - If (SF = 0), indicates the result is positive
  - If the most significant bit (MSB) of the destination operand is set (1), the **SF (1)** is set.

EFLAG-Flag Register

- The Carry flag (**CF**) indicates (unsigned integer overflow).
  - Example, if an instruction has an 8-bit destination operand but the instruction generates a result larger than 11111111 binary, the CF is set.
  - Set (=1) when result is too large (overflow) for destination operand
  - Set (=1) when a larger integer is subtracted from a smaller one

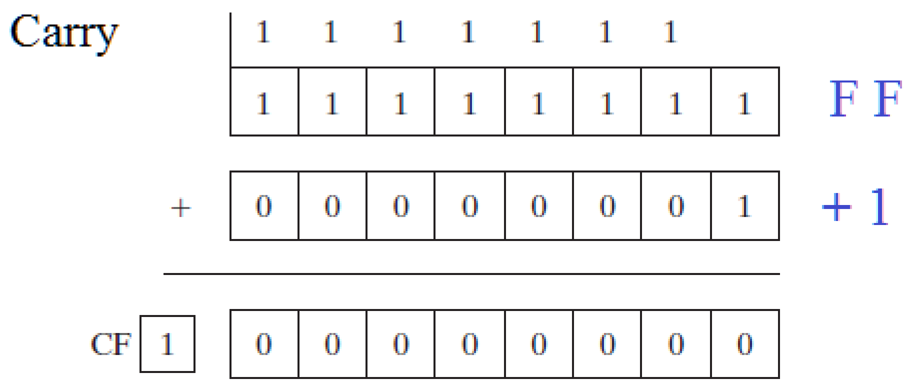
- The Overflow flag (**OF**) indicates (signed integer overflow).
- With signed integer representation, assume MSB is reserved for sign
  - Example: adding 1-byte signed quantity 100d to 1-byte signed quantity 120d will lead to an overflow because 220d > 127d

- CF Example
  CF = 1 when the value exceeds the storage size of its destination operand

  Example, ADD sets the CF because the sum (0xFF) is too large for AL

      mov AL, 0xFF
      add AL,1   ; AL = 00, CF = 1

- The carry out of the highest bit position of AL is copied into the CF



- In the previous example, if 1 added to 0x00FF in AX, the sum fits into 16 bits and CF is clear

```
mov ax,0x00FF
add ax,1               ; AX = 0x0100, CF = 0
```

Understand Overflow

- Generally speaking, overflow occurs when the result of an arithmetic operation generates a resul
  that's "out of range"
- This happens because a register has a limited number of bits, which means that our interpretatio
  of a number comes with a valid range
- For example,
  - adding 1-byte unsigned quantity 240d to 1-byte unsigned quantity 100d will lead to an overflow because 340d > 255d
  - subtracting 1-byte unsigned quantity 240d from 1-byte **unsigned** quantity 100d will lead to an overflow because -140d < 0d (100 – 240 = -140)
  - adding 1-byte signed quantity 100d to 1-byte signed quantity 120d will lead to an overflow because 220d > 127d

Addition/subtraction Instructions

- Addition/subtraction adds/subtracts a value from a destination operand.

  add *destination*, *value*
  sub *destination*, *value*

- Examples

  sub eax, 0x10 Subtracts 0x10 from EAX (EAX = EAX -10)
  add eax, ebx Adds EBX to EAX and stores the result in EAX (EAX = EAX + EBX)
  Source is unchanged (ebx)

- sub an add instructions may modify some flags in the FLAG register:
  CF (Carry Flag)
  OF (Overflow Flag)
  ZF (Zero Flag) (=1 if the result is equal to zero)
  SF (Sign Flag) (=1 if the result is negative)

NOT Instruction

- NOT instruction gets the one's complement of a number by converting each bit to its complement
- Operand can be a register or memory location
- Assume that content of register AL is (1010 0101)
  - NOT AL (AL = 0101 1010)
  - NOT [AL]?

  [AL] is memory address.

  NOT [AL]

  Get the one's complement of a number stored in a memory address [AL]

NOT Instruction Example

NOT AL

NOT	00110011b (AL: 0x33)
Result	11001100b (AL: 0xCC)

NOT [AL+BL]

AL	0x10000000
BL	0x00001234
AL+BL	0x10001234
[AL+BL]	assumed value at memory 0x10001234 is 00001000b
NOT	00001000b
Result	11110111b

NEG Instruction

- NEG (negate) instruction reverses the sign of a number by converting the number to its two's complement
- To get a two's complement of a number, reverse all bits in the destination operand and add 1
  - Assume that content of register AL is 1 (0000 0001)
  - NEG AL (AL = -1, 1111 1111)
  - The Carry, Zero, Sign, Overflow, Auxiliary Carry, and Parity flags are changed according to the value that is placed in the destination operand.

SUB Instruction

- In the sub instruction, the CPU can implement subtraction as a combination of negation and addition.
- Example: The expression 4 - 1 can be rewritten as
  4 + (-1)
  4 + NEG (1)

Carry:	1	1	1	1	1	1				
		0	0	0	0	0	1	0	0	(4)
	+	1	1	1	1	1	1	1	1	(−1)
		0	0	0	0	0	0	1	1	(3)

- The Carry, Zero, Sign, Overflow, Auxiliary Carry, and Parity flags are changed according to the value that is placed in the destination operand.
- CF: Set(=1) when a larger integer is subtracted from a smaller one

- **SF** (Sign Flag)
  - Indicates the result is negative
  - If the most significant bit (MSB) of the destination operand is set, the **SF** is set.
  - Example: mov AL, 1 (AL =1)
    sub AL, 2 (AL 0xFF, SF=1)

	0	0	0	0	0	0	0	1	(1)
+	1	1	1	1	1	1	1	0	(-2)
	1	1	1	1	1	1	1	1	(0xFF)

SF = 1

2:	0000 0010
Reverse bits	1111 1101
+1:	0000 0001
2's complment	1111 1110

To get the correct value

	1	1	1	1	1	1	1	1
	0	0	0	0	0	0	0	0
+1	0	0	0	0	0	0	0	1

Correct value is -1

SF -> (-)

Increment/Decrement Instructions

- The inc /dec increment/decrement a register/memory by one

  inc edx Increments EDX by 1

  dec ecx Decrements ECX by 1

- Example: Assume content of edx is 0x100
  - inc edx (edx = 0x101)
  - mov bx, edx (bx = 0x101)
  - dec bx (bx = 0x100)

Unsigned Integer Division Instruction

The DIV (unsigned divide) instruction performs 32-bit division on unsigned (i.e. positive) integers

Dividend / Divisor = Quotient + Reminder
25 /6 = 4 + 1

Where are we going to save the dividend, divisor, quotient, and reminder values?
How could you perform the division operation?

Unsigned Integer Division Instruction

The DIV (unsigned divide) instruction performs 32-bit division on unsigned integers

```
DIV ECX      ; divide the value stored in (edx:eax) / value stored in (ecx)
             ; save quotient value in EAX
             ; save the reminder in   EDX
```

Dividend	Divisor	Quotient	Reminder
EDX:EAX	register/memory ECX/[ ]	EAX	EDX

Before

EDX	EAX	r/m32(ECX)
0x0	0x7	0x3

DIV ECX

After

EDX	EAX	r/m32(ECX)
0x1	0x2	0x3

Unsigned Integer Division Instruction

The dividend value stored in two registers (EDX:EAX)
EDX: has the signed value (0: positive, 1: negative)
EDX is a sign extension for EAX

Example: Divide 09h/4

```
mov  edx, 0         ; clear edx
mov eax, 0x09
mov ecx, 4
div ecx
```

In this case, the dividend value (+ 0x8003h) is stored in EAX and EDX =0
Devisor value can be save in a register (e.g. ECX) or in a memory location

div ecx ; dividend / divisor

NOT

Dividend	Divisor	Quotient	Reminder
EDX:EAX	register/memory ECX/[ ]	EAX	EDX

Unsigned Integer Division Instruction

EXAMPL: Divide 8003h by 100h

```
mov edx, 0          ; clear edx
mov eax, 8003h      ; dividend value saved in eax
mov ecx, 100h       ; divisor = 100h
div ecx             ; eax = 80h, edx = 3
```

Before

EDX	EAX	r/m32(ECX)
0x0	0x8003h	0x100h

DIV ECX

After

EDX	EAX	r/m32(ECX)
0x3h	0x80h	0x100h

Signed Integer Division Instruction

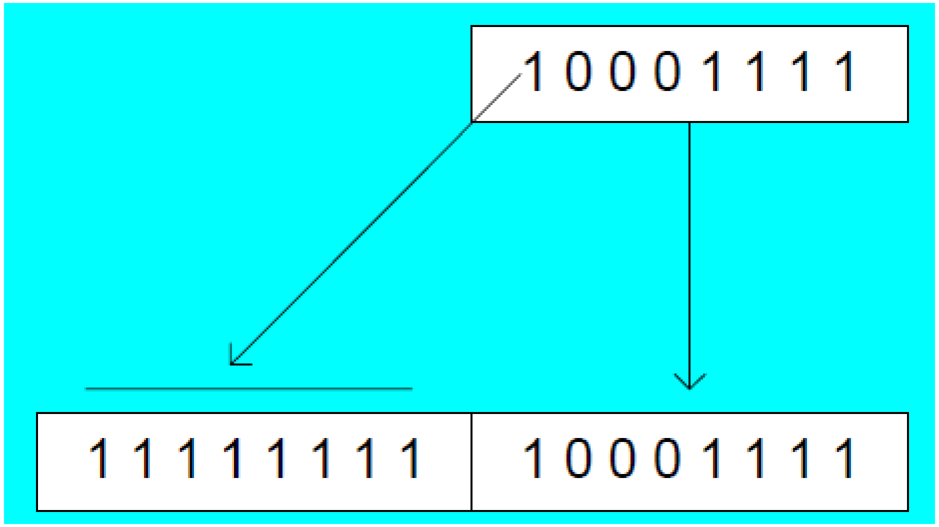
The IDIV (signed divide) instruction performs 32-bit division on signed integers

```
IDIV ECX      ;  divide the value in (edx:eax) / value in (ecx)
              ; edx=1 for negative integer
              ; save quotient value in EAX
              ; save the reminder in   EDX
```

Dividend	Divisor	Quotient	Reminder
EDX:EAX	register/memory ECX/[ ]	EAX	EDX

How could we represent signed integer?

- Signed integers must be sign-extended before division takes place
- Fill high byte with a copy of the sign bit of the low byte
- Fill high word/doubleword with a copy of the sign bit of the low word/ doubleword
- For example, the high byte contains a copy of the sign bit from the low byte:



How could we represent signed integer?

Signed integers must be sign-extended before division takes place

Q: How could you extend signed integer?

Answer: CDQ Instructions

The CBW, CWD, and CDQ instructions provide sign-extension operations:

- CDQ (convert doubleword to quadword) extends EAX into EDX

EXAMPLE:

```
mov eax,  FFFFFF9Bh    ; sign bit  =1
cdq                    ; EDX:EAX = FFFFFFFFFFFFFF9Bh
```

IDIV Example

Example: 32-bit division of (– 48) by 5

```
mov eax,-48
cdq                 ; extend EAX into EDX  (edx = FFFFFFFF)
mov ecx,5
idiv ecx            ; EAX = -9, EDX = -3
```

Example: 32-bit division of (48) by (-5) > (-48/5)

```
mov eax, -48
cdq                    ; extend EAX into EDX  (edx = FFFFFFFF)
mov ecx, 5
idiv ecx               ; EAX = -9, EDX = -3
```

Unsigned Integer Multiplication Instruction

The MUL (unsigned multiply) instruction performs 32-bit multiplication on unsigned (i.e. positive) integers

mul <register or memory reference>

Multiplicand * Multiplier = Product

Multiplicand	Multiplier	Product
EAX	register/memory	EDX:EAX

Save the multiplicand in eax
Multiplier can be saved in a register or memory
The product value will be saved in EDX:EAX
The Carry Flag (CF) indicates the sign of the product
CF = 1: The product is negative (Overflow)
CF = 0: The product is positive

Unsigned Integer Multiplication Instruction

EXAMPLE: Multiply 12345h by 1000h using 3-bit operands
Multiplicand = 12345h (save it in EAX)
Multiplier = 1000h (save in another register. E.g. ebx or ecx)

```
mov eax, 12345h
mov edx, 0     ; clear edx
mov ebx, 1000h
mul ebx
```

Multiplicand	Multiplier	Product
EAX	register/memory	EDX:EAX

NOTE: mul number is not allowed

EDX:EAX = 0000000012345h, CF=0

Before

EDX	EAX	EBX
0x0	12345h	1000h

EDX:EAX = 0000000012345000h

After

EDX	EAX	EBX
0x0000000	12345000h	1000h

Signed Integer Multiplication Instruction

The IMUL (signed multiply) instruction performs 32-bit multiplication on unsigned (i.e. positive) integers

Multiplicand * Multiplier = Product

Multiplicand	Multiplier	Product
EAX	register/memory	EDX:EAX

Save the multiplicand in eax
Multiplier can be saved in a register or memory
The product value will be saved in EDX:EAX (64 bits)
Preserve the sign of the product in EDX
EDX: FFFFFFFF (=1, negative product, CF = 1)
EDX: 000000000 (=0, positive product, CF = 0)
The Overflow flag (**OF**) indicates (signed integer overflow).
If the product value has a negative result greater than -2147483648 decimal, OF = 0
If the product value has a negative result smaller than -2147483648 decimal, OF = 1

Signed Integer Multiplication Instruction

EXAMPLE: Multiply 4823424 by (-423) using 3-bit operands
Multiplicand = 4823424 (save it in EAX)
Multiplier = - 423 (save in another register. E.g. ebx or ecx)

```
mov eax, 4823424
mov ebx, -423
imul ebx
```

Multiplicand	Multiplier	Product
EAX	register/memory	EDX:EAX

EDX:EAX = FFFFFFFF86635D80h, OF=0, CF=0, SF =1

Before

EDX	EAX	EBX
0x0	4823424	- 423

After

EDX	EAX	EBX
FFFFFFFF	86635D80h	- 423

Questions?

Coming Next Week

Bit operations in Assembly Language

Putting It All Together

You should know:

- **Flag Register**
- **Math in Assembly Language**
  - Addition instruction
  - Substation instruction
  - Increment instruction
  - Decrement instruction
  - NOT instruction
  - NEG instruction
  - Unsigned Integer Division
  - Signed Integer Division
  - Unsigned Integer Multiplication
  - Signed Integer Multiplication

Week7 Assignments

- **Learning Materials**

  1- Week 7 Presentation

  2- Read pages: 201-218 in Ch.7: Duntermann, Jeff. Assembly Language Step by Step, Programming with Linux

- **Assignment**

  1- Complete “Lab 7” by coming Sunday 11:59 PM.