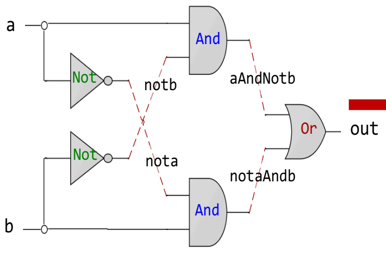
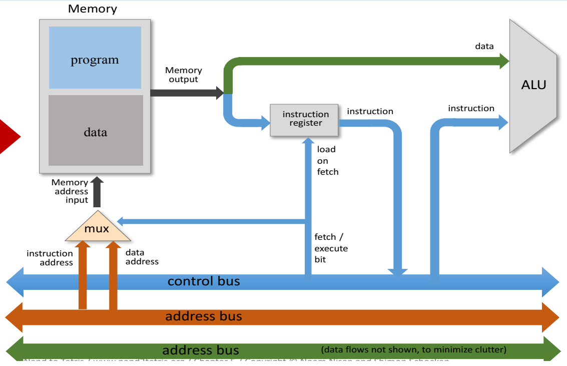
Computer Organization & Assembly Language Programming



```
CHIP Xor {
   IN a, b;
   OUT out;
   PARTS:
   Not(in=a, out=nota);
   Not(in=b, out=notb);
   And(a=nota, b=b, out=w1);
   And(a=a, b=notb, out=w2);
   Or(a=w1, b=w2, out=out);
}
```



```
@R1
D=M
@temp
M=D
```

```
0000000000000001
1111110000010000
0000000000010000
1110001100001000
```

Lecture # 38

Bit Shifting Operations

```
#include<stdio.h>
#include<stdlib.h>
int main(){
  printf("Learning is fun with Arif\n");
  exit(0);
}
```

```
global main
SECTION .data
   msg: db "Learning is fun with Arif", 0Ah, 0h
   len_msg: equ $ - msg
SECTION .text
   main:
      mov rax,1
      mov rdi,1
      mov rsi,msg
      mov rdx,len_msg
      syscall
      mov rax,60
      mov rdi,0
      syscall
```

```
0:  b8 01 00 00 00
5:  bf 01 00 00 00
a:  48 be 00 00 00 00 00
11: 00 00 00
14: ba 1b 00 00 00
19: 0f 05
1b: b8 3c 00 00 00
20: bf 00 00 00 00
25: 0f 05
```

For resources visit my personal website:
https://www.arifbutt.me
and course bitbucket repository:
https://bitbucket.org/arifpucit/coal-repo

Instructor: Muhammad Arif Butt, Ph.D.

Today's Agenda

- Recap: x86-64 Registers, Tool Chain & Instructions
- Shift Operations:
  - SHL/SAL
  - SHR
  - SAR
- Demo (***`bitshift.nasm`***)
- Rotate Operations:
  - ROL
  - ROR
  - RCL
  - RCR
- Demo (***`bitrotate.nasm`***)

Recap

Review: x86-64 Register Set

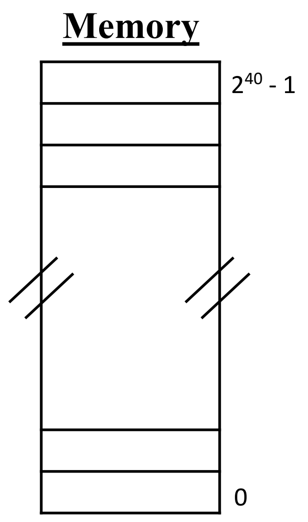
General Purpose Registers

64-bit register	Lowest 32-bits	Lowest 16-bits	Lowest 8-bits
r0/rax	eax	ax	al
r1/rbx	ebx	bx	bl
r2/rcx	ecx	cx	cl
r3/rdx	edx	dx	dl
r4/rsi	esi	si	sil
r5/rdi	edi	di	dil
r6/rbp	ebp	bp	bpl
r7/rsp	esp	sp	spl
r8	r8d	r8w	r8b
r9	r9d	r9w	r9b
r10	r10d	r10w	r10b
r11	r11d	r11w	r11b
r12	r12d	r12w	r12b
r13	r13d	r13w	r13b
r14	r14d	r14w	r14b
r15	r15d	r15w	r15b

SSE Media Registers

511	255	127 0
zmm0	ymm0	xmm0
zmm1	ymm1	xmm1
zmm2	ymm2	xmm2
zmm3	ymm3	xmm3
zmm14	ymm14	xmm14
zmm15	ymm15	xmm15

Memory



	63	0
RIP		EIP

Segment Registers

15 0
CS
DS
SS
ES
FS
GS

FP Registers

79 0
ST0
ST1
ST2
. . .
ST7

	63		21	20	19	18	17	16		14	13	12	11	10	9	8	7	6		4		2		0
RFLAGS	-		ID	VIP	VIF	AC	VM	RF	-	NT	IOP1	IOP0	OF	DF	IF	TF	SF	ZF	-	AF	-	PF	-	CF

Review: x86-64 Tool Chain

first.nasm → Assemble → **first.o** → Link → **myexe** → Load & Execute

first.nasm

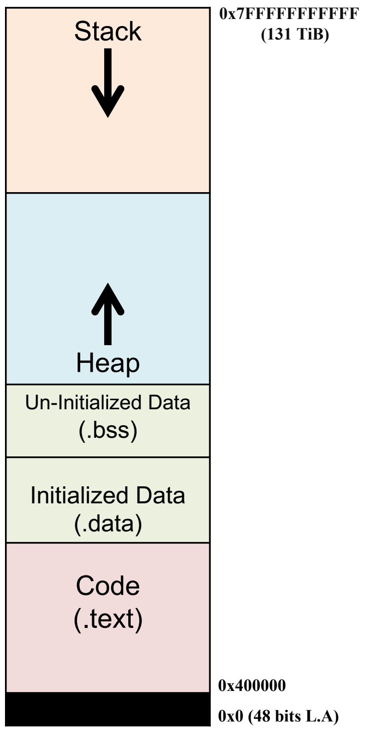
```
; COAL Video Lecture: 30
;  Programmer: Arif Butt
;  first.nasm
SECTION .data
   msg db "Learning…", 0xA
   EXIT_STATUS equ 54
SECTION .bss
;nothing here
SECTION .text
   global _start
   _start:
;display a message on screen
      mov rax,1
      mov rdi,1
      mov rsi,msg
      mov rdx,26
      syscall
;exit the program
      mov rax,60
      mov rdi, EXIT_STATUS
      syscall
```

first.o

```
10001000
01000001
1000101001001001
0101011000011111
0001010011110000
10001000
01001101
10001000
01001001
1000101001001000
0101011000011000
0001010010010001
10001010
01001011
```

myexe

```
1000101001001001
0101011000011111
0001010011110000
10001000
01001101
10001000
10001000
01000001
0101011000011111
0001010011110000
10001000
1000101001001000
0001010010010001
10001010
01001011
0001010011110000
10001000
01001101
10001000
```



- **Processor:** Core 2duo/i3/i5/i7 (64 bit processor)
- **Operating System:** 64 bit Linux Distro (Ubuntu, Kali)
- **Editor:** gedit, vim, atom, sublime, Visual Studio, Eclipse, Xcode
- **Assembler:** NASM, YASM, GAS, MASM
- **Linker:** LD a GNU linker
- **Loader:** Default OS
- **Debugging/RE:** gdb, radare2, objdump and readelf

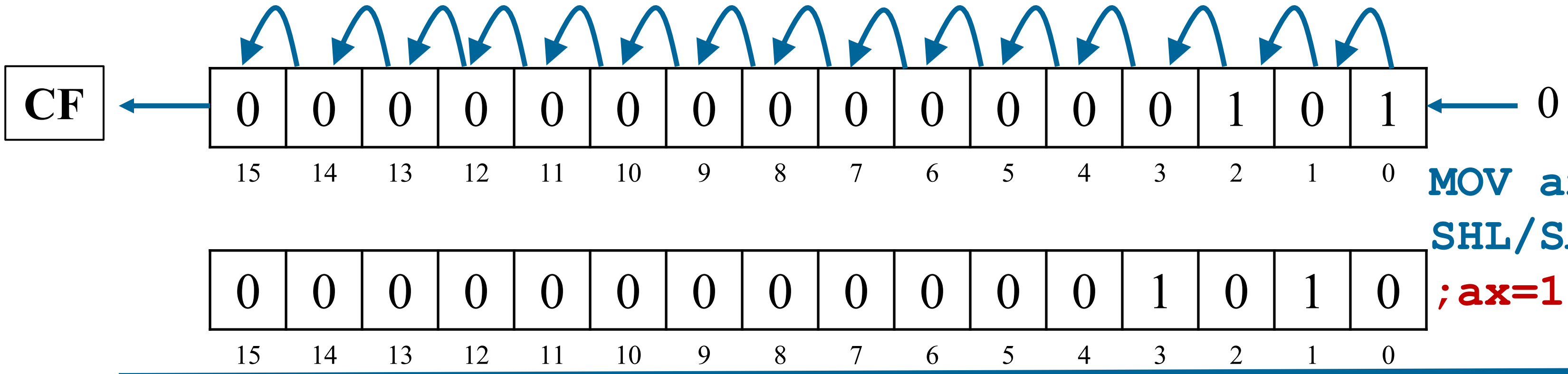
Review: Categories of x86-64 Instructions

Category	Description	Examples
Data Transfer	Move from source to destination	`mov, movzx, movsx, lea, lds, lss, xchg, push, pop, pusha, popa, pushf, popf`
Arithmetic	Arithmetic on integer	`add, addc, sub, subb, mul, imul, div, idiv, neg, inc, dec, cmp`
Bit Manipulation	Logical & bit shifting operations	`and, or, not, xor, test, shl/sal, shr, sar, ror, rol, rcr, rcl`
Control Transfer	Conditional and unconditional jumps, and procedure calls	`jmp` `jcc(jz,jnz,jg,jge,jl,jle,jc,jnc,...)` `call, ret`
String	Move, compare, input and output	`movsb, movsw, lodsb, lodsw, stosb, stosw, rep, repz, repe, repnz, repne`
Floating Point	Arithmetic	`fld, fst, fstp, fadd, fsub, fmul, fdiv`
Conversion	Data type conversions	`cbw, cwd, cdq, xlat`
Input Output	For input and output	`in, out`
Miscellaneous	Manipulate individual flags	`clc, stc, cld, std, sti`

Shift Instructions

Logical/Arithmetic Shift Left (SHL/SAL)

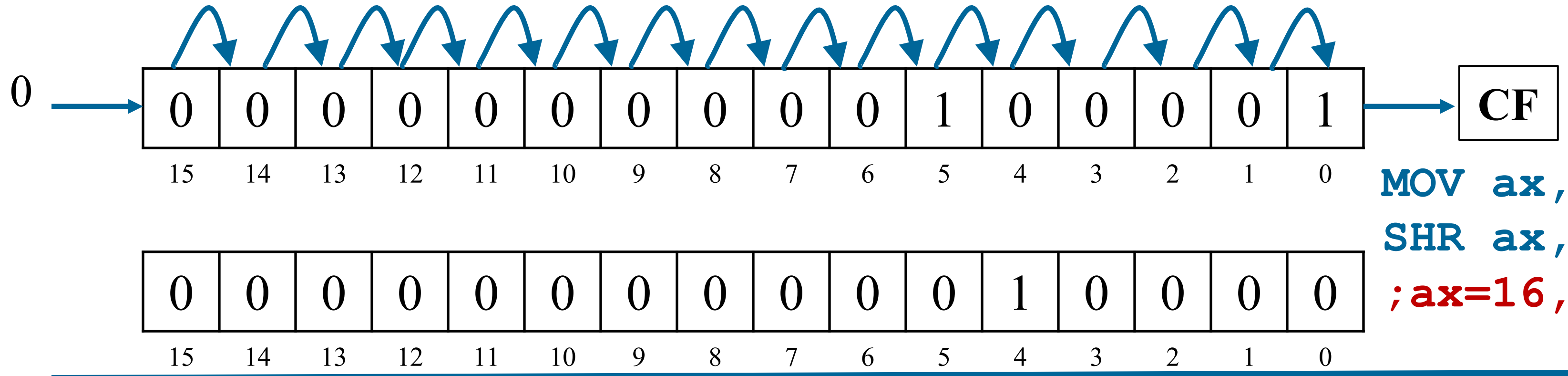
- **Format:** `SHL/SAL dest, count`
- **Operation:** Shifts the bits in the destination to the **left** by count bits. A **zero** is pushed into the least significand bit position and the **msb** is shifted into the CF
- **Operands:** Destination operand can be a reg/mem
  Count (<=63) operand can be an immediate value or CL
- **Flags Affected:** The CF contains the last most significand bit shifted out of the destination operand
- **Usage:** Used to multiply the **signed/unsigned** destination contents with 2^n, where n is the number of bits shifted



```
MOV ax, 5d
SHL/SAL ax, 1
;ax=10, cf=0
```

Logical Shift Right (SHR) Instruction

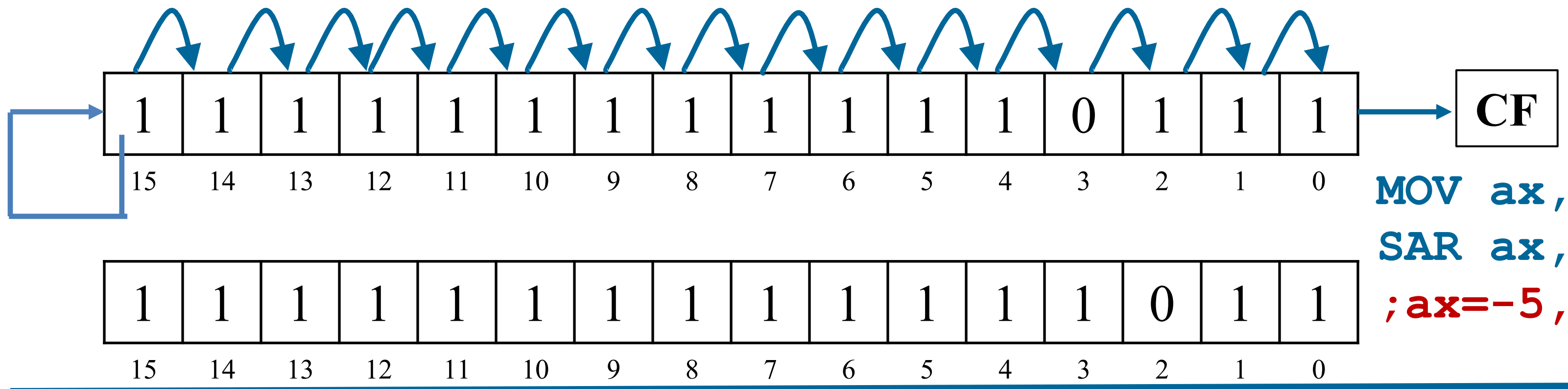
- **Format:** `SHR dest, count`
- **Operation:** Shifts the bits in the destination to the **right** by count bits. A **zero** is pushed into the most significand bit position and the **lsb** is shifted into the CF
- **Operands:** Destination operand can be a reg/mem
  Count (<=63) operand can be an immediate value or CL
- **Flags Affected:** The CF contains the last least significand bit shifted out of the destination operand
- **Usage:** Used to divide the **unsigned** destination contents with 2^n, where n is the number of bits shifted



```
MOV ax, 33d
SHR ax, 1
;ax=16, cf=1
```

Shift Arithmetic Right (SAR) Instruction

- **Format:** `SAR dest, count`
- **Operation:** Shifts the bits in the destination to the **right** by count bits. The **sign bit** is pushed into the most significand bit position and the **lsb** is shifted into the CF
- **Operands:** Destination operand can be a reg/mem
  Count (<=63) operand can be an immediate value or CL
- **Flags Affected:** The CF contains the last least significand bit shifted out of the destination operand
- **Usage:** Used to divide the **signed/unsigned** destination contents with 2^n, where n is the number of bits shifted

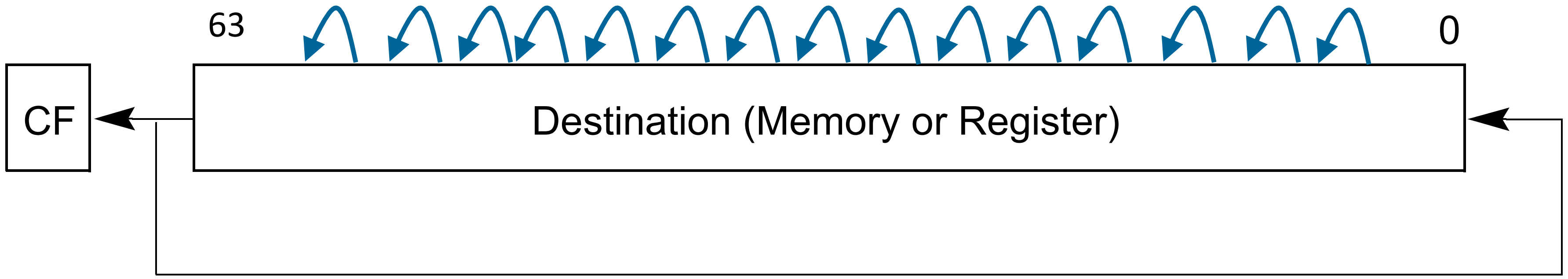


```
MOV ax, -9d
SAR ax, 1
;ax=-5, cf=1
```

Assembling & Executing x86-64 Program

Rotate Instructions

Rotate Left (ROL) Instruction

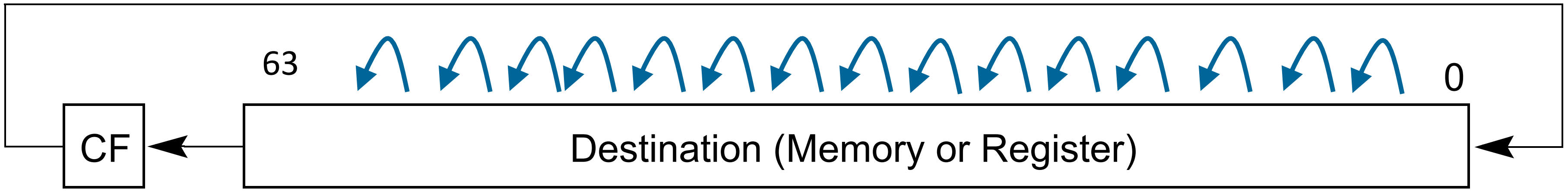


- **Format:** `ROL dest, count`
- **Operation:** The **msb** is placed into the **CF** as well as pushed into the **lsb**. The remaining bits are moved one position to the left. This is performed count number of times
- **Operands:** Destination operand can be a reg/mem

  Count (<=63) operand can be an immediate value or CL
- **Flags Affected:** The CF contains the last most significand bit shifted out of the destination operand
- **Usage:** Used for bit shifts across multiple words

```
mov al, 11110000b
rol al, 1
;al=11100001 (e1)
;cf=1
```

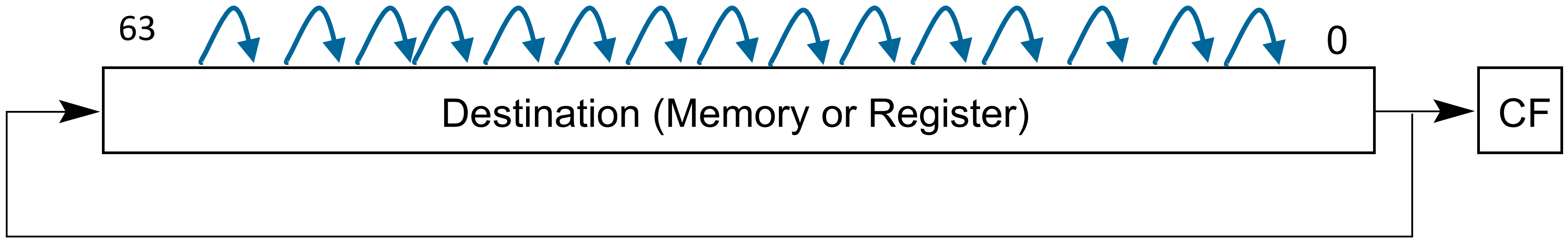
Rotate Carry Left (RCL) Instruction



- **Format:** `RCL dest, count`
- **Operation:** The previous value of the **CF** is shifted into the **lsb** and the remaining bits are moved one position to the left. Finally, the **msb** is removed and placed in the **CF**. This is performed count number of times
- **Operands:** Destination operand can be a reg/mem
  Count (<=63) operand can be an immediate value or CL
- **Flags Affected:** The CF contains the last most significand bit shifted out of the destination operand
- **Usage:** Used for bit shifts across multiple words

```
clc          ; cf=0
mov al, 88h  ; al=10001000 (88h)
rcl al, 1    ; al=00010000 (10h), cf=1
rcl al, 1    ; al=00100001 (21h), cf=0
```

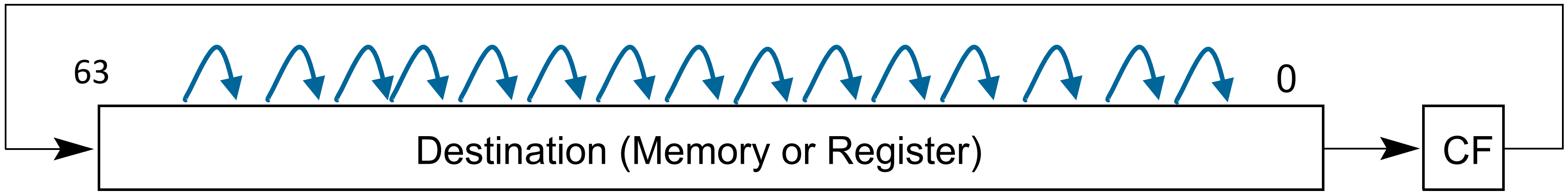
Rotate Right (ROR) Instruction



- **Format:** ROR dest, count
- **Operation:** The **lsb** is placed into the **CF** as well as pushed into the **msb**. The remaining bits are moved one position to the right. This is performed count number of times
- **Operands:** Destination operand can be a reg/mem
  Count (<=63) operand can be an immediate value or CL
- **Flags Affected:** The CF contains the last least significand bit shifted out of the destination operand
- **Usage:** Used for bit shifts across multiple words

```
mov al, 11110000b
ror al, 1
;al=01111000 (78h)
;cf=0
```

Rotate Carry Right (RCR) Instruction



- **Format:** `RCR dest, count`
- **Operation:** The previous value of the **CF** is shifted into the **msb** and the remaining bits are moved one position to the right. Finally, the **lsb** is removed and placed in the **CF**. This is performed count number of times
- **Operands:** Destination operand can be a reg/mem

  Count (<=63) operand can be an immediate value or CL
- **Flags Affected:** The CF contains the last least significand bit shifted out of the destination operand
- **Usage:** Used for bit shifts across multiple words

```
stc            ; cf=1
mov al, 10h ; al=00010000 (10h)
rcr al, 1    ; al=10001000 (88h), cf=0
```

Assembling & Executing x86-64 Program

Things To Do

Coming to office hours does NOT mean you are academically week!