Announcement

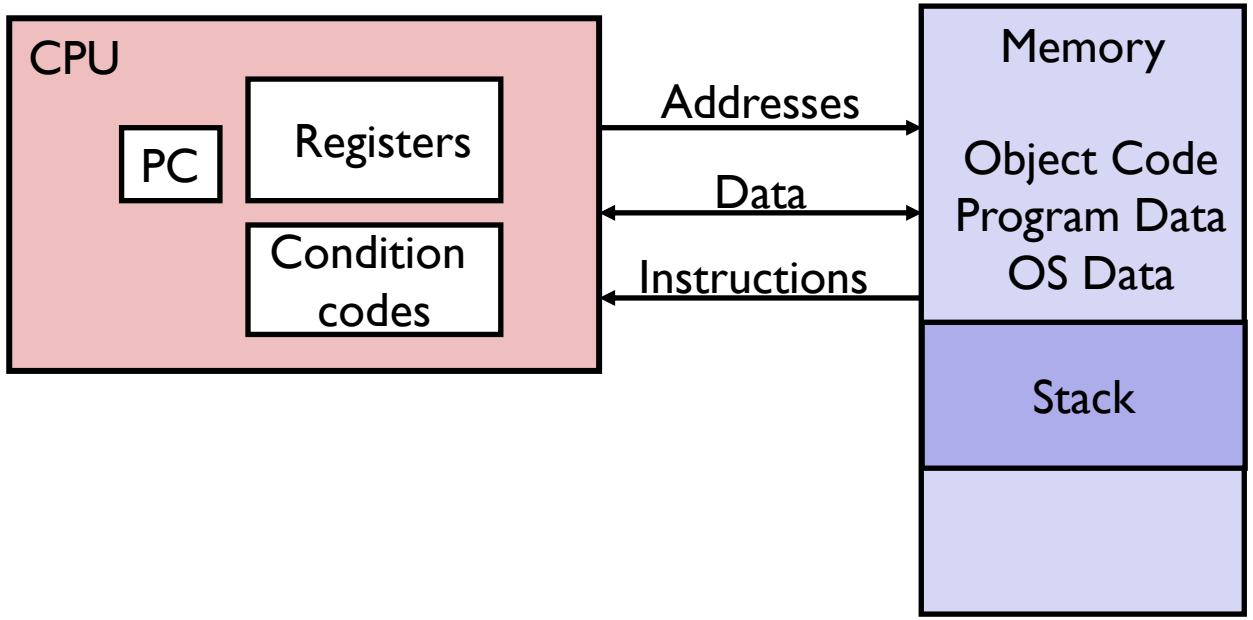
- Assignment 2
  - Due October 10
  - Building an accumulator-based system – Part I
  - Compile and run your program on a lab machine before submitting it.

Lecture 14

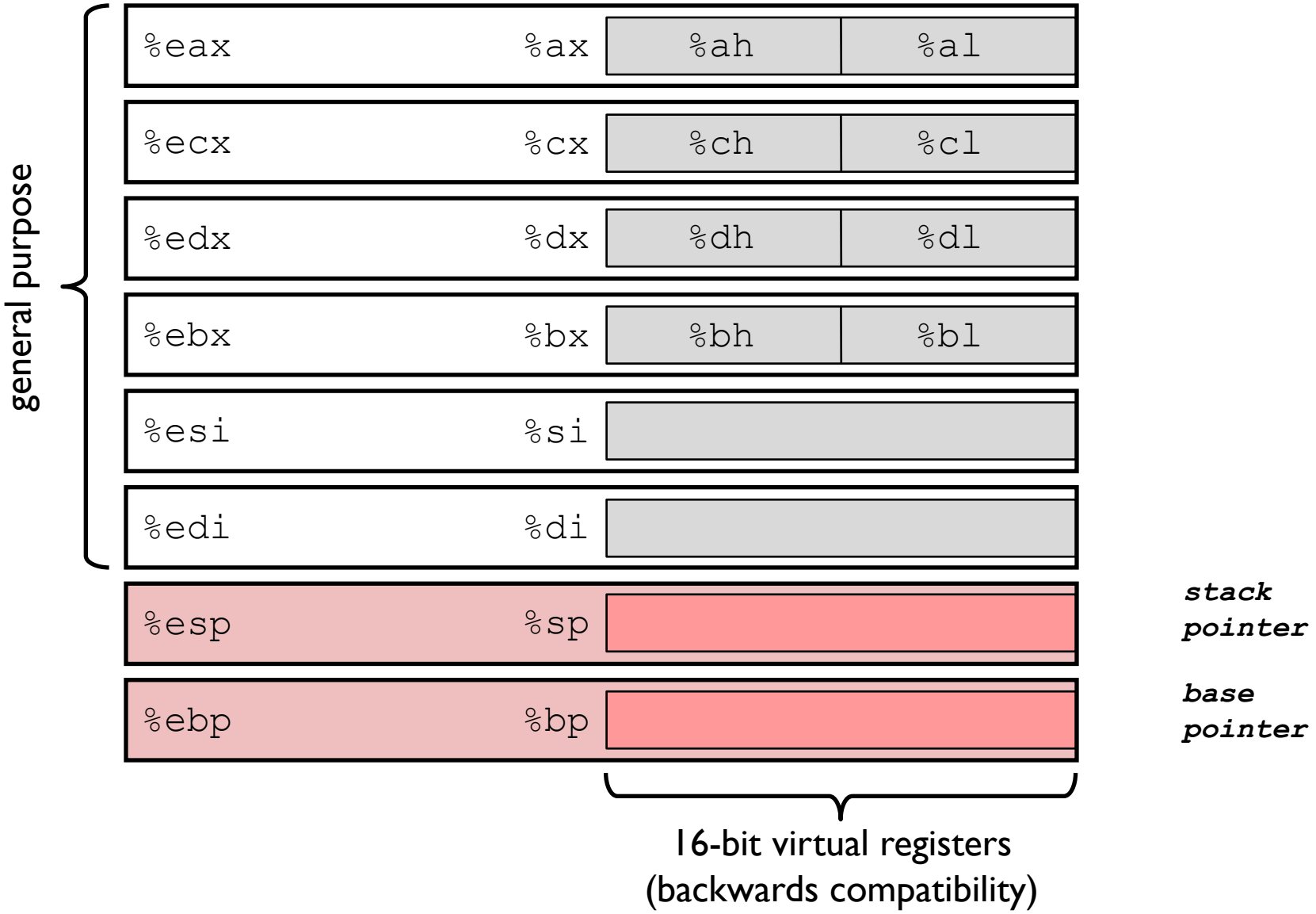
Basic Instructions in IA-32

CPSC 275
Introduction to Computer Systems

Programmer's View on Computer System



Integer Registers



Assembly Characteristics: Data Types

- Integral data of 1, 2, or 4 bytes
  - Data values
  - Addresses
- Floating point data of 4, 8, 10, or 12 bytes (more on this later)
- No aggregate types such as arrays or structures
  - Just contiguously allocated bytes in memory

Assembly Characteristics: Operations

- Perform arithmetic function on *register* or *memory* data

- Transfer data between memory and register
  - Load data from memory into register
  - Store register data into memory

- Transfer control
  - Unconditional jumps to/from procedures
  - Conditional branches

Data Formats (IA-32)

- *word* : 16-bit data type

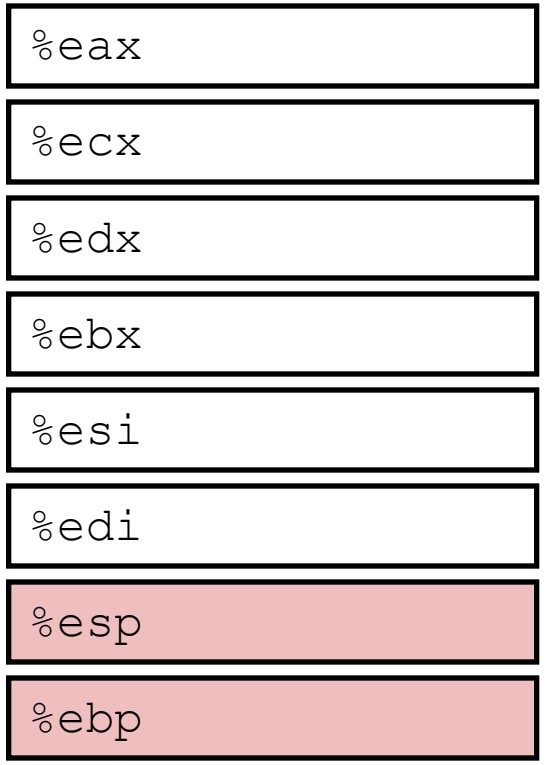
C types	Intel data types	Assembly suffix	Size (# bytes)
`char`	byte	`b`	1
`short`	word	`w`	2
`int`	double word	`l`	4

Moving Data

- Moving Data

  `movl` *source*, *dest*

- Operand Types
  - ***Immediate:*** Constant integer data
    - Example: **`$0x400`**, **`$-533`**
    - Like C constant, but prefixed with **`'$'`**
    - Encoded with 1, 2, or 4 bytes
  - ***Register:*** One of 8 integer registers
    - Example: **`%eax`**, **`%edx`**
    - But **`%esp`** and **`%ebp`** reserved for special use
    - Others have special uses for particular instructions
  - ***Memory:*** consecutive memory at address given by register
    - Simplest example: **`(%eax)`**
    - Various other "address modes"



`movl` Operand Combinations

	Source	Dest	Src,Dest	C Analog
movl	*Imm*	*Reg*	`movl $0x4,%eax`	`temp = 0x4;`
		Mem	`movl $-147,(%eax)`	`*p = -147;`
	Reg	*Reg*	`movl %eax,%edx`	`temp2 = temp1;`
		Mem	`movl %eax,(%edx)`	`*p = temp;`
	Mem	*Reg*	`movl (%eax),%edx`	`temp = *p;`

Cannot do memory-memory transfer with a single instruction

Memory Addressing Modes

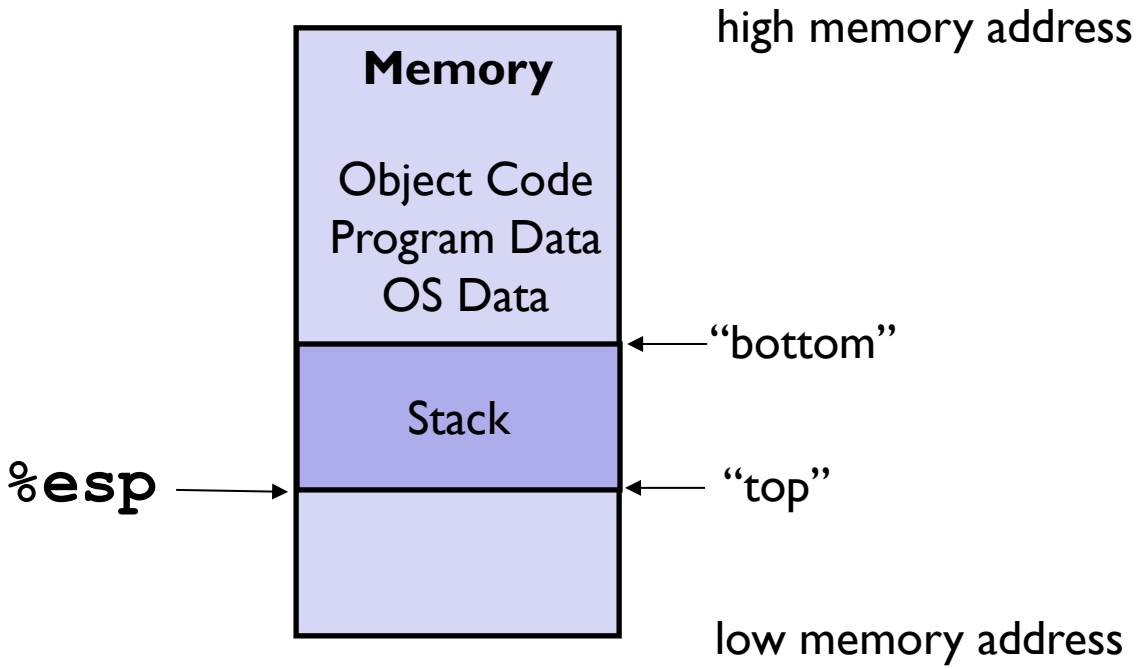
- Normal: (R) Mem[Reg[R]]
  - Register R specifies memory address

```
movl (%ecx),%eax
```

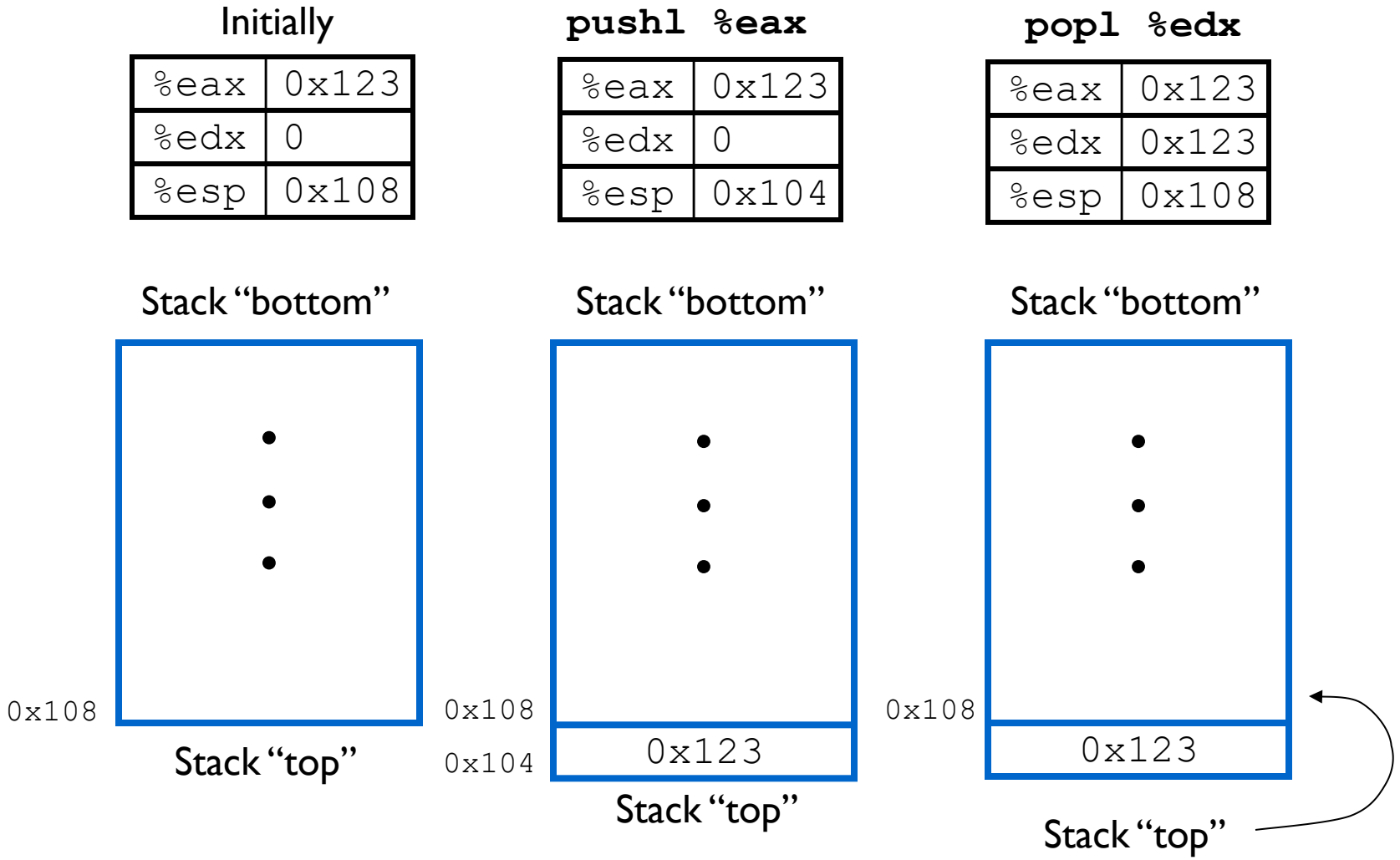
- With displacement: D(R) Mem[Reg[R]+D]
  - Register R specifies start of memory region
  - Constant displacement D specifies offset

```
movl 8(%ebp),%edx
```

The Stack Pointer: `%esp`

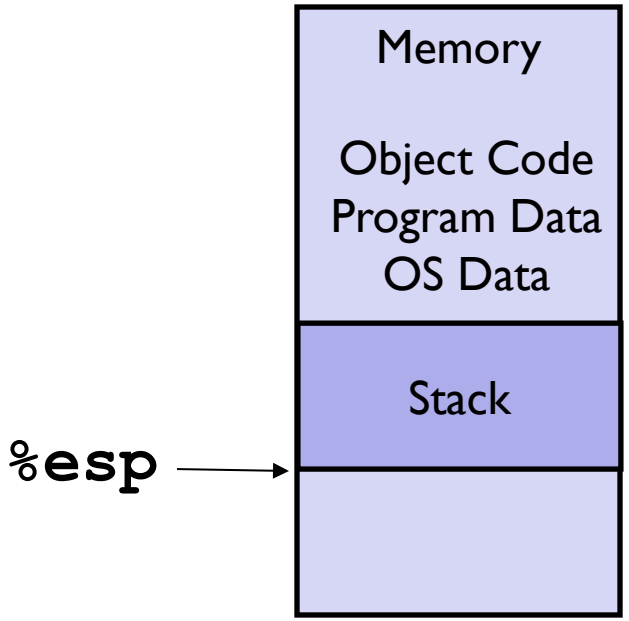


Moving Data to/from Stack



`pushl` and `popl`

```
pushl src
popl  dest
```




```
pushl %eax
```

is equivalent to:

```
subl $4, %esp
movl %eax, (%esp)
```

```
popl %ecx
```

is equivalent to:

```
movl (%esp), %ecx
addl $4, %esp
```

Other Memory Addressing Modes

- Most General Form

  D(Rb,Ri,S) Mem[Reg[Rb]+S*Reg[Ri]+ D]

  - D: Constant "displacement" (any immediate values)
  - Rb: Base register: Any of 16 integer registers
  - Ri: Index register: Any, except for **`%esp`**
  - S: Scale: 1, 2, 4, or 8

- Special Cases

  (Rb,Ri) Mem[Reg[Rb]+Reg[Ri]]

  D(Rb,Ri) Mem[Reg[Rb]+Reg[Ri]+D]

  (Rb,Ri,S) Mem[Reg[Rb]+S*Reg[Ri]]

Address Computation Examples

`%edx`	`0xf000`
`%ecx`	`0x0100`

Expression	Address Computation	Address
`0x8 (%edx)`		

Address Computation Examples

`%edx`	`0xf000`
`%ecx`	`0x0100`

Expression	Address Computation	Address
`0x8(%edx)`	`0xf000 + 0x8`	`0xf008`

Address Computation Examples

`%edx`	`0xf000`
`%ecx`	`0x0100`

Expression	Address Computation	Address
`0x8(%edx)`	`0xf000 + 0x8`	`0xf008`
`(%edx,%ecx)`		

Address Computation Examples

%edx	0xf000
%ecx	0x0100

Expression	Address Computation	Address
0x8(%edx)	0xf000 + 0x8	0xf008
(%edx,%ecx)	0xf000 + 0x100	0xf100

Address Computation Examples

`%edx`	`0xf000`
`%ecx`	`0x0100`

Expression	Address Computation	Address
`0x8(%edx)`	`0xf000 + 0x8`	`0xf008`
`(%edx,%ecx)`	`0xf000 + 0x100`	`0xf100`
`(%edx,%ecx,4)`		

Address Computation Examples

%edx	0xf000
%ecx	0x0100

Expression	Address Computation	Address
0x8(%edx)	0xf000 + 0x8	0xf008
(%edx,%ecx)	0xf000 + 0x100	0xf100
(%edx,%ecx,4)	0xf000 + 4*0x100	0xf400

Address Computation Examples

%edx	0xf000
%ecx	0x0100

Expression	Address Computation	Address
0x8(%edx)	0xf000 + 0x8	0xf008
(%edx,%ecx)	0xf000 + 0x100	0xf100
(%edx,%ecx,4)	0xf000 + 4*0x100	0xf400
0x80(,%edx,2)		

Address Computation Examples

%edx	0xf000
%ecx	0x0100

Expression	Address Computation	Address
0x8(%edx)	0xf000 + 0x8	0xf008
(%edx,%ecx)	0xf000 + 0x100	0xf100
(%edx,%ecx,4)	0xf000 + 4*0x100	0xf400
0x80(,%edx,2)	2*0xf000 + 0x80	0x1e080

Shift Instructions

Format		Computation	
`sall`	*Src,Dest*	Dest = Dest << Src	# also called `shll`
`sarl`	*Src,Dest*	Dest = Dest >> Src	# arithmetic
`shrl`	*Src,Dest*	Dest = Dest >> Src	# logical

- Here, *src* is the shift amount given either as an immediate or in the single byte register `%cl`.
- Can take either one or two arguments. If only one is supplied, the number of bits to shift is one. For example,

```
shrl $1,%eax      is equivalent to
shrl %eax
```