Data Transfer Instructions

Muhammad Afzaal
m.afzaal@nu.edu.pk

Book Chapter

- “Assembly Language for x86 Processors”
- Author “Kip R. Irvine”
- 6th Edition
- Chapter 4
  - Section 4.1

Operand Types (1/2)

- We know the format of an instruction from previous lectures

  `[label:] mnemonic [operand] [;comment]`

- Instructions can have 0 – 3 operands
- Operands can be any of
  - Register: Name of an x86 register
  - Memory: Reference to a memory location
  - Immediate value: a numeric literal

Direct Memory Operands

- Name of a variable shows its offset in data segment

```
.data
var DB 10h
.code
mov al, var
```

- If `var` is stored at address `00000001h`, when assembled, above code produces following machine instruction

```
A0 00000001
```

MOV Instruction (1/2)

- Copies data from a source operand to a destination operand
- First operand is the destination and second operand is the source

```
MOV dest, src
```

MOV Instruction (2/2)

- Some rules to follow when using `MOV`
  - Both operands must have same size
  - Both operands cannot be memory operands
  - `CS`, `EIP`, `IP` cannot be destination operands
  - An immediate value cannot be moved to a segment register
- Some useful variants of `MOV`

```
MOV reg, reg
MOV reg, mem
MOV reg, imm
MOV mem, reg
MOV mem, imm
```

Zero/Sign Extension of Integers (1/4)

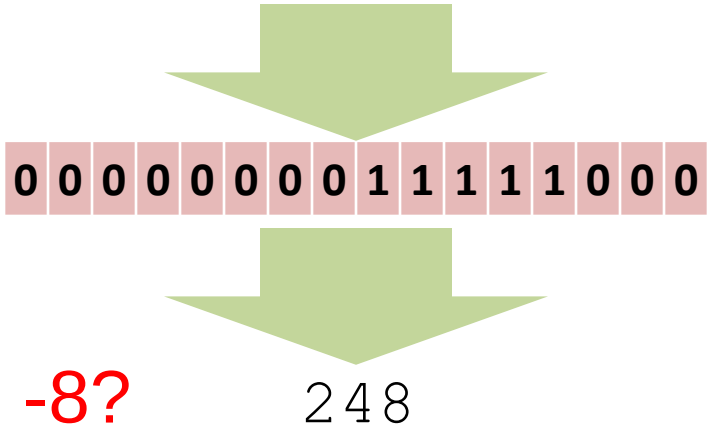
- `MOV` cannot copy data directly from a smaller operand to a larger one
- Suppose we want to move a *byte* variable `var` into a 16-bit register `ax`

```
.data
var DB 10h
.code
MOV ax, 0
MOV al, var
```

Zero/Sign Extension of Integers (2/4)

- What happens if same approach is followed to copy a negative number?
- What is the value in `ax` after this code is assembled?

```
.data
svar DB -8
.code
MOV ax, 0
MOV al, svar
```

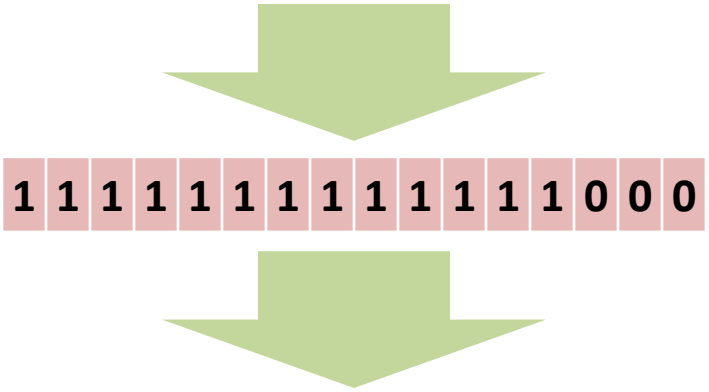


- What happened to -8?

Zero/Sign Extension of Integers (3/4)

- How about doing like this?

```
.data
svar DB -8
.code
MOV ax, 0FFFFFFFFh
MOV al, svar
```



-8

Zero/Sign Extension of Integers (4/4)

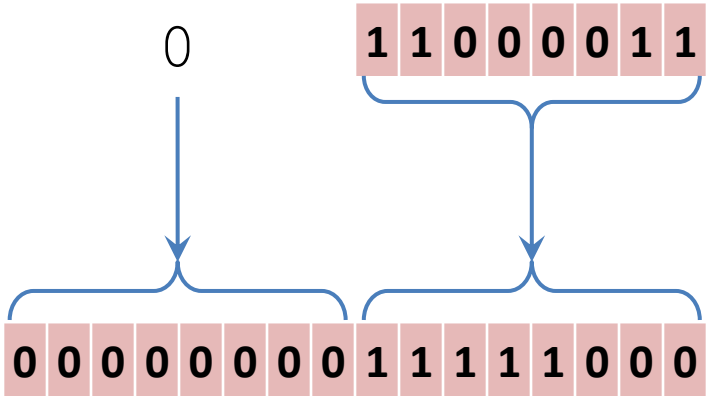
- These examples show different approaches for signed and unsigned number
  - In case of unsigned numbers, a zero is extended to all higher order bits of the destination operand
  - In case of signed numbers, the sign-bit is extended to all higher order bits of the destination operand

MOVZX Instruction (1/2)

- `MOVZX` (MOVe with Zero-eXtend) copies the source operand into destination operand and extends zeroes in the remaining higher order bits of destination operand
- It has three variants
  - `MOVZX reg32, reg/mem8`
  - `MOVZX reg32, reg/mem16`
  - `MOVZX reg16, reg/mem8`

MOVZX Instruction (2/2)

```
.data
val DB 11000011b
.code
MOVZX ax, val
```

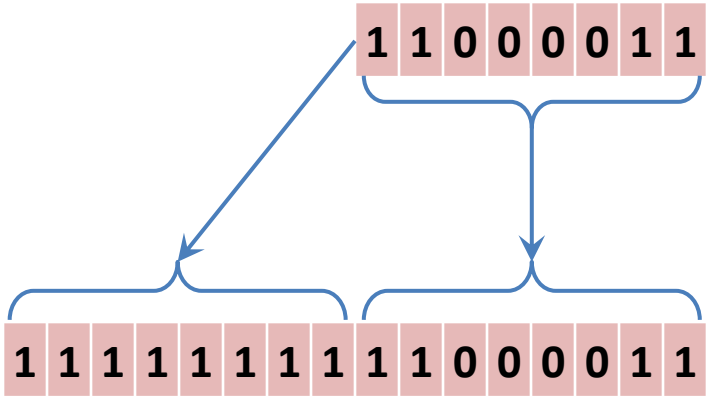


MOVSX Instruction (1/2)

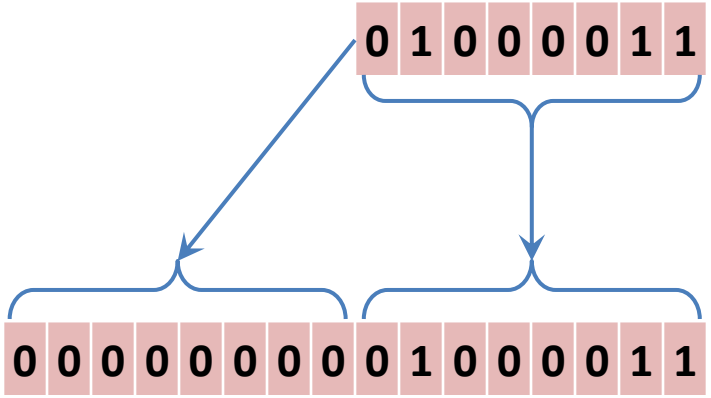
- `MOVSX` (MOVe with Sign-eXtend) copies the source operand into destination operand and extends the sign-bit in remaining higher order bits in destination operand
- It has three formats
  - `MOVSX reg32, reg/mem8`
  - `MOVSX reg32, reg/mem16`
  - `MOVSX reg16, reg/mem8`

MOVSX Instruction (2/2)

```
.data
val DB 11000011b
.code
MOVSX ax, val
```




```
.data
val DB 01000011b
.code
MOVSX ax, val
```



LAHF and SAHD Instructions

- `LAHF` (Load `AH` from status Flags) instruction copies lower byte of EFLAGS register into `AH`
- Sign, Zero, Auxiliary Carry, Parity and Carry flags are copied
- `SAHF` (Store `AH` into status Flags) instruction copies `AH` into lower byte of EFLAGS register

XCHG Instruction

- `XCHG` instruction exchanges the contents of two operands
- This instruction has three different variants
  - `XCHG` reg, reg
  - `XCHG` reg, mem
  - `XCHG` mem, reg
- To exchange two memory operands, a register is used to as temporary container

```
XCHG al, ah
```

Direct-Offset Operands (1/2)

- Adding a displacement or offset to the name of a variable
- This technique makes it possible to access memory locations which do not have explicit labels
- For example, to access individual elements of an array

Direct-Offset Operands (2/2)

```
.data
arr DB 1, 2, 3, 4, 5
.code
MOV al, arr
MOV ah, arr+1
```

```
.data
arr DW 1, 2, 3, 4, 5
.code
MOV al, arr
MOV ah, arr+2
```