Computer Systems Organisation (CS2.201)

Summer 2021, IIIT Hyderabad

07 June, Monday (Lecture 7) – Data Movement Instructions

Taught by Prof. Avinash Sharma

Single-Suffix Instructions

They are generally used to copy data from one location to another, but cannot be used to move between two memory locations. A register must be involved. Further, an immediate value cannot be the second operand.

Instruction		Effect	Description
MOV	*S*, *D*	*D* ← *S*	Move
`movb`		Move byte	
`movw`		Move word	
`movl`		Move double word	
MOVS	*S*, *D*	*D* ← SignExtend(*S*)	Move with sign extension

Figure 1: Data Movement Instructions

Three-suffix Instructions

The character suffixes on `mov` indicate the:

- extension method: `z` for zero-extended and `s` for sign-extended
- source data size: `b` for byte, `w` for word and `l` for double word
- destination location size: as for source.

Instruction		Effect	Description
`movsbw`		Move sign-extended byte to word	
`movsbl`		Move sign-extended byte to double word	
`movswl`		Move sign-extended word to double word	
MOVZ	*S*, *D*	*D* ← ZeroExtend(*S*)	Move with zero extension
`movzbw`		Move zero-extended byte to word	
`movzbl`		Move zero-extended byte to double word	
`movzwl`		Move zero-extended word to double word	

Figure 2: Three-suffix Opcodes

When the latter two are identical, we only specify one and leave out the extension method, as in Figure 1.

In x86-64, for sign-extension we have `movsbq`, `movswq` and `movslq` analogously. In the case of zero-extension, however, we only need `movzbq` and `movzwq`, since copying from 4 to 8 bytes automatically extends it with zeroes.

Stack Operations

The `push` instruction takes an argument `S` and is equivalent to

```
R[%esp] <- R[%esp] - 4
M[R[%esp]] <- S
```

The `%esp` (stack pointer) register points to the memory location which is the "top" of the stack, but in reality addresses increase towards the base of the stack. Therefore we decrement the address in order to push.

The `pop` instruction, similarly, takes an argument `D` and is equivalent to

```
D <- M[R[%esp]]
R[%esp] <- R[%esp] + 4
```

Example

Consider the C code:

```
int exchange (int *xp, int y)
{
    int x = *xp;
    *xp = y;
    return x;
}
```

It compiles to

```
-- xp at %ebp+8 and y at %ebp+12
movl 8(%ebp), %edx  [Get xp]
movl (%edx), %eax   [x = *xp, becomes return value]
movl 12(%ebp), %ecx [Get y]
movl %ecx, (%edx)   [Store y at xp]
```