

Short Notes Chapter 2 [Lecture 5-7]

Assembly Directives:

1. Define byte 'db'

Syntax: db somevalue

Use: used for single byte

2. Define word 'dw'

Syntax: dw somevalue

Use: used for two bytes

Label:

A symbol associated to a point in the program is called a label and is written as the label name followed by a colon (:)

Labels are used to initialize constants as well as variables with re-loadable addresses.

Direct Addressing:

In direct addressing the memory address is fixed and is given in the instruction. The actual data used is placed in memory and now that data can be used as the destination operand as well. Also, the source and destination operands must have the same size.

Method to access memory is direct addressing, whether it is the MOV instruction or the ADD instruction.

Legal/Illegal instructions:

None of the general operations of mov add, sub etc. allow moving data from memory to memory.

Only register to register, register to memory, memory to register, constant to memory, and constant to register operations are allowed.

The other register to constant, memory to constant, and memory to memory is all disallowed.

Only string instructions allow moving data from memory to memory

Ambiguous instructions:

The instruction "mov [num1], 5" is legal but there is no way for the processor to know the data movement size in this operation. The variable num1 can be treated as a byte or as a word and similarly 5 can be

treated as a byte or as a word. Such instructions are declared ambiguous by the assembler. The assembler has no way to guess the intent of the programmer. And memory is a linear array and label is an address in it. There is no size associated with a label.

Therefore, to resolve its ambiguity we clearly tell our intent to the assembler in one of the following ways.

mov byte [num1], 5

mov word [num1], 5

Register Indirect addressing:

There is another method in which the address can be placed in a register so that it can be changed is called Register indirect addressing. In this addressing, a register is used to reference memory.

Registers that can hold the address of data:

There are four registers in iAPX88 architecture that can hold address of data and they are BX, BP, SI, and DI.

In iapx88 architecture brackets can be used around BX, BP, SI, and DI only.

Effective address:

All the addressing mechanisms in iAPX88 return a number called effective address. For example, in base + offset addressing, neither the base nor the offset alone tells the desired cell in memory to be accessed. It is only after the addition is done that the processor knows which cell to be accessed. This number which came as the result of addition is called the effective address.

Address Wraparound:

In Physical address calculation a carry if generated is dropped without being stored anywhere, for example BX=0100, DS=FFF0 and the access under consideration is [bx+0x0100]. The effective address will be 0200 and the physical address will be 100100. This is a 21bit answer and cannot be sent on the address bus which is 20 bits wide. The carry is dropped and just like the segment wraparound our physical memory has wrapped around at its very top.

Segment Wraparound:

Segment wraparound occurs when during the effective address calculation, a carry is generated. This carry is dropped giving the effect that when we try to access beyond the segment limit, we are actually wrapped around to the first cell in the segment.

For example, if BX=9100, DS=1500 and the access is [bx+0x7000] we form the effective address 9100 + 7000 = 10100. The carry generated is dropped forming the actual effective address of 0100.

Addressing Modes:

The iAPX88 processor supports seven modes of memory access.

Direct

A fixed offset is given in brackets and the memory at that offset is accessed.

For example, “mov [1234], ax” stores the contents of the AX registers in two bytes starting at address 1234 in the current data segment. The instruction “mov [1234], al” stores the contents of the AL register in the byte at offset 1234.

Based Register Indirect

A base register is used in brackets and the actual address accessed depends on the value contained in that register.

For example, “mov [bx], ax” moves the two-byte contents of the AX register to the address contained in the BX register in the current data segment. The instruction “mov [bp], al” moves the one-byte content of the AL register to the address contained in the BP register in the current stack segment.

Indexed Register Indirect

An index register is used in brackets and the actual address accessed depends on the value contained in that register.

For example, “mov [si], ax” moves the contents of the AX register to the word starting at address contained in SI in the current data segment. The instruction “mov [di], ax” moves the word contained in AX to the offset stored in DI in the current data segment.

Based Register Indirect + Offset

A base register is used with a constant offset in this addressing mode. The value contained in the base register is added with the constant offset to get the effective address.

For example, “mov [bx+300], ax” stores the word contained in AX at the offset attained by adding 300 to BX in the current data segment. The instruction “mov [bp+300], ax” stores the word in AX to the offset attained by adding 300 to BP in the current stack segment.

Indexed Register Indirect + Offset

An index register is used with a constant offset in this addressing mode. The value contained in the index register is added with the constant offset to get the effective address.

For example, “mov [si+300], ax” moves the word contained in AX to the offset attained by adding 300 to SI in the current data segment and the instruction “mov [di+300], al” moves the byte contained in AL to the offset attained by adding 300 to DI in the current data segment.

Base + Index

One base and one index register is used in this addressing mode. The value of the base register and the index register are added together to get the effective address.

For example, “mov [bx+si], ax” moves the word contained in the AX register to offset attained by adding BX and SI in the current data segment. The instruction “mov [bp+di], al” moves the byte contained in AL to the offset attained by adding BP and DI in the current stack segment. Other examples are “mov [bx+di], ax” and “mov [bp+si], ax.”

Base + Index + Offset

This is the most complex addressing method and is relatively infrequently used. A base register, an index register, and a constant offset are all used in this addressing mode. The values of the base register, the index register, and the constant offset are all added together to get the effective address.

For example, “mov [bx+si+300], ax” moves the word contents of the AX register to the word in memory starting at offset attained by adding BX, SI, and 300 in the current data segment.