Assembly language programming
By xorpd

Basic Assembly

Basic Memory Instructions

xorpd.net

Objectives

- We will learn about the following:
  - How to access memory using the x86 instructions.
  - How to count addresses properly.
  - How to store a dword in memory: Which byte comes first?
  - Advanced addressing with the brackets syntax.
  - Some limitations when accessing memory in the x86 processor.

The brackets [ ]

- The brackets are the usual syntax for **accessing memory**.
  - [x] represents the contents of the cell in address x.

x-4	x-3	x-2	x-1	x	x+1	x+2	x+3	x+4
0xa	0x3	0x0	0x0	0x25	0xff	0xff	0x11	0x12

- The brackets could be used with most of the instructions that we have learned about.
  - (Although there are some limitations)

- Examples:

```
add      eax,dword [ecx]
movzx    eax,word [some_mem]
neg      dword [esi]
xor      dword [edi],esi
```

Addressing

- The Byte (8 bits) is the basic quantity regarding x86 memory management.
  - You can not read or write less than one byte.
  - All the addresses count bytes.

- You can use the dword, word, byte operators to specify the wanted size of memory read/write.
  - If you don't specify, the assembler will complain.
  - Examples:
    - `mov    [eax],3         ; Will not assemble.`
    - `mov    dword [eax],3`
    - `add    word [eax],3`
    - `sub    byte [eax],3`

Endianness

http://farm3.staticflickr.com/2021/2060860569_64701ea497_o.jpg

- There is more than one way to store a dword (4 bytes) in memory.
- In x86 processors, when a dword is stored to memory, the least significant byte is stored in the lowest address.
  - Little Endian / The intel convention.
- In some other processors, the dword is stored such that the least significant byte is stored in the highest address.
  - Big Endian.
- Example: Storing the dword 0x12345678 in memory:
  - Little Endian (x86):

...	402000	402001	402002	402003	...
…	78	56	34	12	…

  - Big Endian:

...	402000	402001	402002	402003	...
…	12	34	56	78	…

Example

- The assembler can not guess your perception about memory.
- Example:
  - You want to store 3 consecutive dwords in memory.
  - First attempt:

```
section '.data' data readable writeable
          my_dwords    dd 3 dup (0)

section '.text' code readable executable
start:
          mov        esi,my_dwords
          mov        dword [esi],1
          inc        esi
          mov        dword [esi],2
          inc        esi
          mov        dword [esi],3
```

Example (Cont.)

- The assembler can not guess your perception about memory.
- Example:
  - You want to store 3 consecutive dwords in memory.
  - First attempt:

```
section '.data' data readable writeable
           my_dwords     dd 3 dup (0)

section '.text' code readable executable
start:
       →   mov        esi,my_dwords
           mov        dword [esi],1
           inc        esi
           mov        dword [esi],2
           inc        esi
           mov        dword [esi],3
```

0	1	2	3	4	5	6	7	8	9	a	b

Example (Cont.)

- The assembler can not guess your perception about memory.
- Example:
  - You want to store 3 consecutive dwords in memory.
  - First attempt:

```
section '.data' data readable writeable
          my_dwords      dd 3 dup (0)

section '.text' code readable executable
start:
          mov         esi,my_dwords
      →   mov         dword [esi],1
          inc         esi
          mov         dword [esi],2
          inc         esi
          mov         dword [esi],3
```

0	1	2	3	4	5	6	7	8	9	a	b
00	00	00	00	00	00	00	00	00	00	00	00

esi ↑

Example (Cont.)

- The assembler can not guess your perception about memory.
- Example:
  - You want to store 3 consecutive dwords in memory.
  - First attempt:

```
section '.data' data readable writeable
          my_dwords      dd 3 dup (0)

section '.text' code readable executable
start:
          mov        esi,my_dwords
          mov        dword [esi],1
       →  inc        esi
          mov        dword [esi],2
          inc        esi
          mov        dword [esi],3
```

0	1	2	3	4	5	6	7	8	9	a	b
01	00	00	00	00	00	00	00	00	00	00	00

↑

Example (Cont.)

- The assembler can not guess your perception about memory.
- Example:
  - You want to store 3 consecutive dwords in memory.
  - First attempt:

```
section '.data' data readable writeable
          my_dwords     dd 3 dup (0)

section '.text' code readable executable
start:
          mov        esi,my_dwords
          mov        dword [esi],1
          inc        esi
       →  mov        dword [esi],2
          inc        esi
          mov        dword [esi],3
```

0	1	2	3	4	5	6	7	8	9	a	b
01	00	00	00	00	00	00	00	00	00	00	00
	↑										

Example (Cont.)

- The assembler can not guess your perception about memory.
- Example:
  - You want to store 3 consecutive dwords in memory.
  - First attempt:

```
section '.data' data readable writeable
          my_dwords     dd 3 dup (0)

section '.text' code readable executable
start:
          mov        esi,my_dwords
          mov        dword [esi],1
          inc        esi
          mov        dword [esi],2
       →  inc        esi
          mov        dword [esi],3
```

0	1	2	3	4	5	6	7	8	9	a	b
01	02	00	00	00	00	00	00	00	00	00	00
	↑										

Example (Cont.)

- The assembler can not guess your perception about memory.
- Example:
  - You want to store 3 consecutive dwords in memory.
  - First attempt:

```
section '.data' data readable writeable
           my_dwords      dd 3 dup (0)

section '.text' code readable executable
start:
           mov         esi,my_dwords
           mov         dword [esi],1
           inc         esi
           mov         dword [esi],2
           inc         esi
        →  mov         dword [esi],3
```

0	1	2	3	4	5	6	7	8	9	a	b
01	02	00	00	00	00	00	00	00	00	00	00
		↑									

Example (Cont.)

- The assembler can not guess your perception about memory.
- Example:
  - You want to store 3 consecutive dwords in memory.
  - First attempt:

```
section '.data' data readable writeable
          my_dwords      dd 3 dup (0)

section '.text' code readable executable
start:
          mov        esi,my_dwords
          mov        dword [esi],1
          inc        esi
          mov        dword [esi],2
          inc        esi
          mov        dword [esi],3
```

$\rightarrow$

0	1	2	3	4	5	6	7	8	9	a	b
01	02	03	00	00	00	00	00	00	00	00	00

$\uparrow$

Example (Cont.)

- The assembler can not guess your perception about memory.
- Example:
  - You want to store 3 consecutive dwords in memory.
  - First attempt:

```
section '.data' data readable writeable
          my_dwords     dd 3 dup (0)

section '.text' code readable executable
start:
          mov        esi,my_dwords
          mov        dword [esi],1
          inc        esi
          mov        dword [esi],2
          inc        esi
          mov        dword [esi],3
```

  - Not the result we wanted…

Example (Cont.)

- The assembler can not guess your perception about memory.
- Example:
  - You want to store 3 consecutive dwords in memory.
  - The correct way:

```
section '.data' data readable writeable
          my_dwords     dd 3 dup (0)

section '.text' code readable executable
start:
          mov        esi,my_dwords
          mov        dword [esi],1
          add        esi,4
          mov        dword [esi],2
          add        esi,4
          mov        dword [esi],3
```

0	1	2	3	4	5	6	7	8	9	a	b
01	00	00	00	02	00	00	00	30	00	00	00

Example (Cont.)

- The assembler can not guess your perception about memory.
- Example:
  - You want to store 3 consecutive dwords in memory.
  - The correct way:

```
section '.data' data readable writeable
          my_dwords     dd 3 dup (0)

section '.text' code readable executable
start:
          mov         esi,my_dwords
          mov         dword [esi],1
          add         esi,4
          mov         dword [esi],2
          add         esi,4
          mov         dword [esi],3
```

- Remember: **x86 Addresses always count bytes.**

Advanced addressing

- You could put more complicated expressions inside the brackets.
- Examples:
  - **`mov dword [ecx + 1],3`**
  - **`sub byte  [ecx + esi],3`**
  - **`neg word  [ecx + esi*2]`**
  - **`add dword [ecx + esi*2 + 3],4`**

- There is a limit to the possible complexity.
- These will not assemble:
  - `mov dword [ecx + esi + edi],3`
  - `mov dword [ecx*177],4`

Memory to memory limitation

- The following will not assemble:
  - `mov      dword [eax],dword [edx]`
  - `xor      dword [eax],dword [ecx]`

- x86 can handle at most one memory argument at a time.
  - (There are exceptions though).

Summary

- Brackets are the generic syntax for memory access in x86 assembly.
- Addresses count bytes.
- x86 uses the Little Endian convention. (Least significant byte in lowest address)
- The brackets support advanced addressing.
  - But they have their limits.
- You usually can’t copy memory to memory directly using an x86 instruction.