CSD1100

Assembler - Arrays

Vadim Surov

Allocation multiple elements

- `lb1 db 0, 1, 2, 3`
  - Defines 4 bytes, initialized to 0, 1, 2 and 3
  - `lb1` is a label of the first byte
- `lb2 db “w“, “o“, ‘r‘, ‘d‘, 0`
  - Defines a null-terminated string, initialized to “word\0”
  - `lb2` is a label of the beginning of the string
- `lb3 db “word“, 0`
  - Same as above, but more convenient to use

Allocation with the times qualifier

- Let's say you want to declare 100 bytes all initialized to 0
- NASM provides a nice way to do this by using **the times qualifier**

```
lb3 times 100 db 0
```

Init value of each byte in the sequence

equivalent to

```
bl3 db 0,0,0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,0,0,
        ... (100 times 0s)
```

db and friends

- db, dw, dd, dq, dt, dq, and do are used to declare initialized data in the output file.
- d - data (1 byte), w - word (2 bytes), d - double word (4 bytes), q - quadro word (8 bytes)
- They can be invoked in a wide range of ways

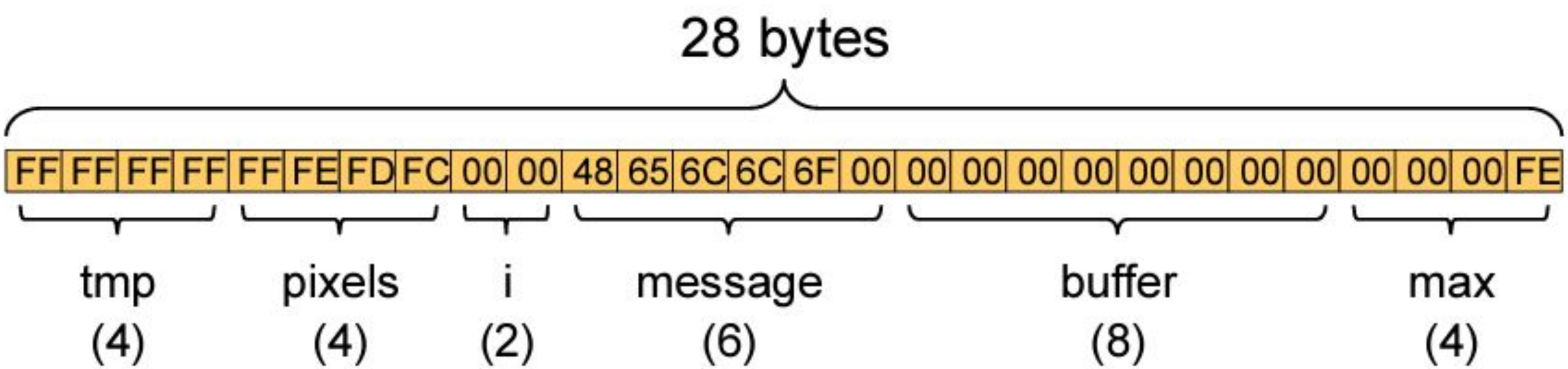
db and friends

db.asm

```
section .data
        db      0x55                    ; just the byte 0x55
        db      0x55,0x56,0x57          ; three bytes in succession
        db      'a',0x55                ; character constants are OK
        db      'hello',13,10,'$'       ; so are string constants
        dw      0x1234                  ; 0x34 0x12
        dw      'a'                     ; 0x41 0x00 (it's just a number)
        dw      'ab'                    ; 0x41 0x42 (character constant)
        dw      'abc'                   ; 0x41 0x42 0x43 0x00 (string)
        dd      0x12345678              ; 0x78 0x56 0x34 0x12
        dq      0x1122334455667788      ; 0x88 0x77 0x66 0x55 0x44 0x33 0x22 0x11
        dd      1.234567e20             ; floating-point constant
        dq      1.234567e20             ; double-precision float
        dt      1.234567e20             ; extended-precision float
```

Data layout in memory

```
tmp        dd     -1
pixels     db     0FFh, 0FEh, 0FDh, 0FCh
i          dw     0
message    db     "H", "e", "llo", 0
buffer     times  8    db  0
max        dd     254
```



Effective addressing

- The effective address is **the location of a value in memory** defined as an operand of the instruction.
  - Note: only one operand can be effective address
- nasm has a very simple syntax for it: consists of an expression evaluating to the desired address, enclosed in square brackets [ ].
- For example:

```
b db 0, 1, 2, 3, 4, 5, 6, 7
mov rax, [b+1]    ; move 1 to rax
```

Effective addressing

- nasm is capable of doing calculations of quite complex expressions on effective addresses using labels, registers and constants.
- For example:
    - `mov rax, [b+(rcx-1)*8+1]`
- Some expressions are not allowed though:
    - `(1-rax)`, for example, is not allowed

mov instruction size

- When size of operation data cannot be deducted by nasm (ex. when destination of `mov` is a memory), size of moving data must be specified.
- For example:
  - `mov byte [rax+1], 'A'`
  - `cmp word [rdi], 0`
  - `cmp dword [rdi+rcx], 10`
  - `cmp qword [rdi-8], 20`

Example Str

```
; Str.
; Set elements of array (as a null-terminated string)
; Run: $ nasm -f elf64 str.asm && ld -dynamic-linker /li
; Output: ABC

%include "macros.inc"

section .data
arr times 10 db 0    ; reserve 10 bytes and fill with 0
fmt db "%s",10,0

section .text
    global _start
    extern printf

_start:
    mov      rax, arr
    mov byte [rax], 'A'
    mov byte [rax+1], 'B'
    mov byte [rax+2], 'C'

    PRINTF fmt, arr
    EXIT
```

Example Str

```
(gdb) run
Starting program: /mnt/c/Users/vadim/OneDrive/Desktop/Pr

Breakpoint 1, _start () at str.asm:17
17              mov         rax, arr
(gdb) p/d (char[10]) arr
$1 = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
(gdb) s
18              mov byte [rax], 'A'
(gdb) s
19              mov byte [rax+1], 'B'
(gdb) s
20              mov byte [rax+2], 'C'
(gdb) p/d (char[10]) arr
$2 = {65, 66, 0, 0, 0, 0, 0, 0, 0, 0}
(gdb) p/c (char[10]) arr
$3 = {65 'A', 66 'B', 0 '\000', 0 '\000', 0 '\000', 0 '\
(gdb)
```

Example Copy

copy.asm

```
; Copy string.
; Copy 24 bytes from src to dst by using nasm's effective
; Run:
; $ nasm -f elf64 copy.asm && ld -dynamic-linker /lib64/l
; Output:
; 012345678901234567890123

%include "macros.inc"

section .data
src     db '0123456789012345678901234567890123456789',10,0
dst     times 30 db 0
fmt     db "%s",10,0
```

Example Copy

copy.asm

```
; Copy string.
; Copy 24 bytes from src to dst by using nasm's effective
; Run:
; $ nasm -f elf64 copy.asm && ld -dynamic-linker /lib64/l
; Output:
; 012345678901234567890123

%include "macros.inc"

section .data
src     db '0123456789012345678901234567890123456789',10,0
dst     times 30 db 0
fmt     db "%s",10,0
```

Example Copy

```
section .text
    global _start
    extern printf

_start:
    mov rcx, 3
repeat:
    ; src -/-> dst, so src --> reg --> dst
    mov rax, [src+(rcx-1)*8]
    mov [dst+(rcx-1)*8], rax
    loop repeat

    PRINTF fmt, dst
    EXIT
```

References

1. NASM documentation
   https://www.nasm.us/doc/