Symbolic Constants, Data Related Operators and Directives

Muhammad Afzaal
m.afzaal@nu.edu.pk

Book Chapter

- “Assembly Language for x86 Processors”
- Author “Kip R. Irvine”
- 6th Edition
- Chapter 3
  - Section 3.5

Symbolic Constants

- Created by associating an identifier with an integer expression or text
- Identifier also called symbol
- Different from a variable
  - **Symbols do not reserve storage**
  - **Symbols cannot change at runtime**
- Symbols can be created by using equal-sign directive, EQU and TEXTEQU directives

Equal-Sign Directive

- Associates a symbol name with an integer expression

```
name = expression
```

- At assemble time, all occurrences of `name` are replaced by `expression`
- Helpful when an expression is used many times in a program

EQU Directive

- Associates a symbolic name with an integer expression or some arbitrary text
- Three different formats
  - *name* EQU *expression*
  - *name* EQU *symbol*
  - *name* EQU <*text*>
- `expression` must be a valid integer expression
- `symbol` is an existing symbol name
- `text` can be any string

Some Useful Symbols

- Current Location Counter
  - `$` gives the address of location where used

```
self_ptr DW $
```

- Keyboard Definitions
  - Numeric keys can also be used by defining symbols

```
esc_key = 27
mov al, esc_key
```

Calculating Size of Arrays and Strings

- Size of array/string can be calculated with the help of location counter
  - `arr DB 1, 2, 3, 4, 5`
  - `arr_size = ($ - arr)`

Book Chapter

- “Assembly Language for x86 Processors”
- Author “Kip R. Irvine”
- 6th Edition
- Chapter 4
  - Section 4.3

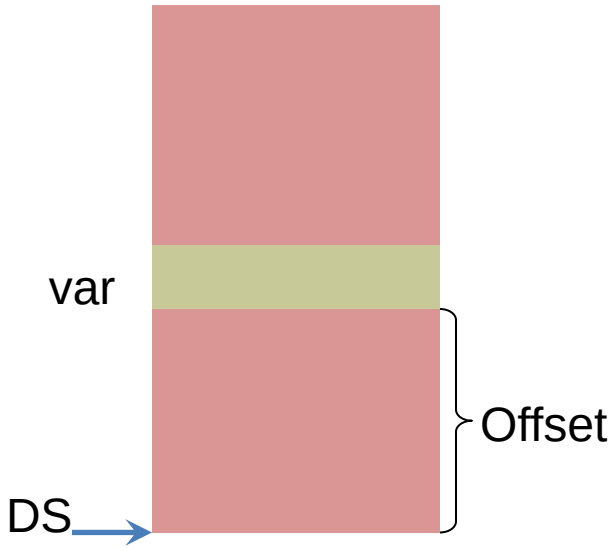
Operators and Directives

- Operators and Directives are not executable instructions
- Different MASM operators and directives to get information about the addresses and size of data
  - OFFSET
  - PTR
  - TYPE
  - LENGTHOF
  - SIZEOF

OFFSET Operator (1/2)

- This operator returns the offset of a data label
- Offset is the distance in bytes, of label from the beginning of data segment

```
.data
…
var DB 10
…
```



OFFSET Operator (2/2)

```
.data
…
var DB 10
…

.code
…
mov esi, OFFSET var
…
```

PTR Operator (1/2)

- Sometimes source and destination in an assembly instruction have different types
- How to handle this situation?
- `PTR` moves first `X` bytes from `Y` into `dest`

```
mov dest, X PTR Y
```

  - `X` is an assembly directive like `DB`, `DW` etc.
  - `Y` is source operand

PTR Operator (2/2)

- x86 processors store data in little endian format
- So lower addressed byte in memory contains least significant byte of data
- `PTR` moves the first byte of `var` into `al` which is `80h` in this case

```
.data
var DW 9A80h
.code
mov al, DB PTR var
```

TYPE Operator

- Returns the size of a single element of a variable in bytes

```
.data
var1 DB 9Ah
var2 DD 9A80h

TYPE var1   → Returns 1
TYPE var2   → Returns 4
```

LENGTHOF Operator

- Returns the number of elements in array or string appearing on the first line

```
.data
arr DB 9Ah, 0Ch, 81h
str DB “Hello”,0
```

```
LENGTHOF arr    → Returns 3
LENGTHOF str    → Returns 6
```

SIZEOF Operator

- Returns total size in bytes of a variable, array or string
- Total size is obtained by multiplying `TYPE` with `LENGTHOF` values of a variable or array

```
.data
arr DW 109Ah, 6B0Ch, 2681h
str DB “Hello World!”,0

SIZEOF arr      → Returns 6
SIZEOF str      → Returns 13
```

LABEL Directive

- Can be used to give a group name to some variables
- Inserts a label with a size attribute without allocating any storage

```
.data
lab LABEL DB
var DW 1234h
```

```
.code
mov al, lab
```

`al` contains `34h`

Why `34h`, and not `12h`?

ALIGN Directive (1/2)

- Used to align a variable on 1, 2, 4 or 16 bytes
- Why align?
  - Because CPU can process data stored at even-numbered addresses more quickly

```
.data                Address
a DB ?          → 0000
b DW ?          → 0001
c DB ?          → 0002
d DD ?          → 0003
```

ALIGN Directive (2/2)

- How to align previous data on even addresses?

```
.data          Address
a DB ?         → 0000
ALIGN 2
b DW ?         → 0002
c DB ?         → 0004
ALIGN 2
d DD ?         → 0006
```

Calculate Addresses…

- Write down the addresses of these data

```
.data                Address
a DB ?           → 0000
ALIGN 2
b DB ?           →
ALIGN 4
c DW ?           →
ALIGN 2
d DD ?           →
ALIGN 8
e DW ?           →
```