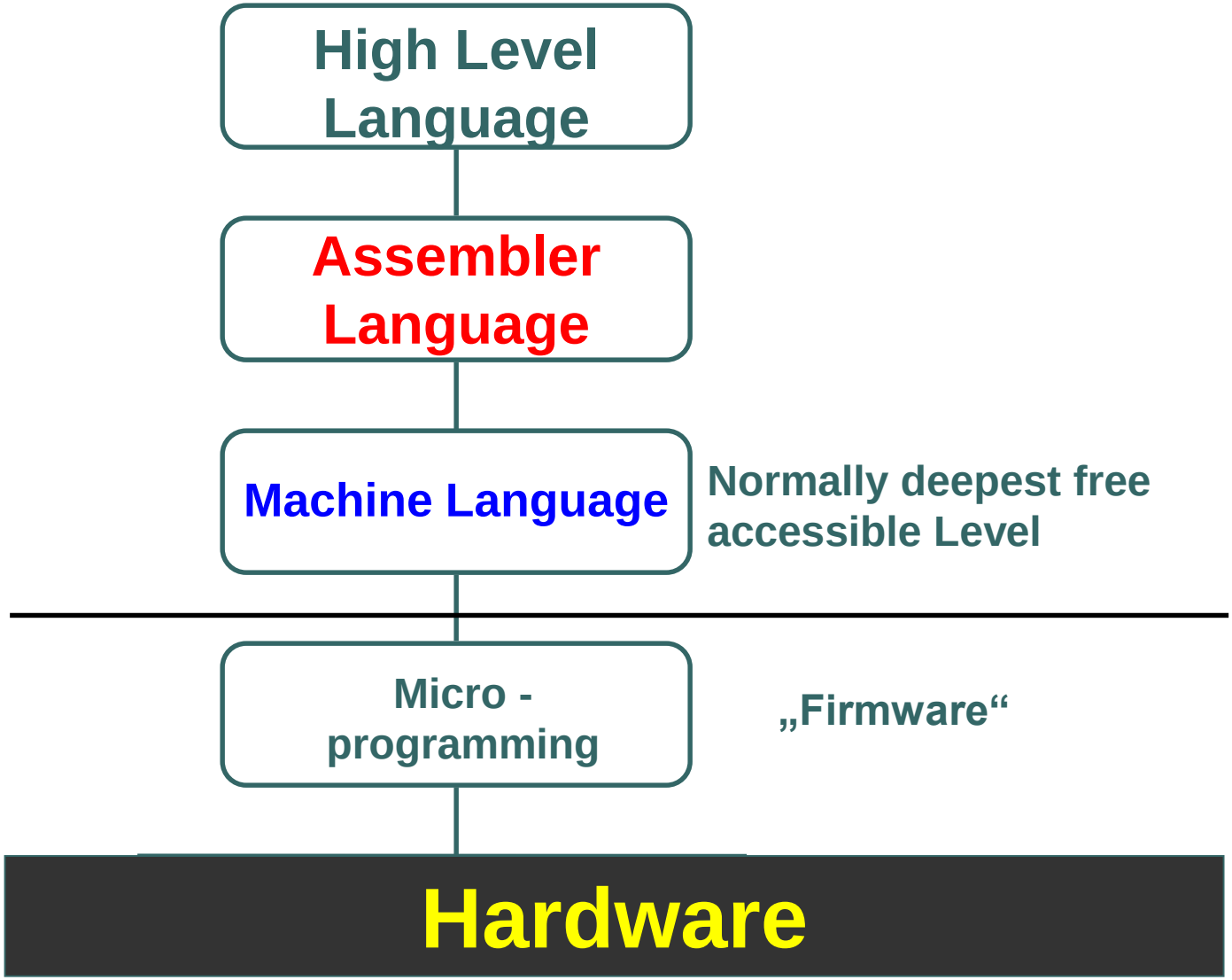
Assembler

A short overview

Language Levels

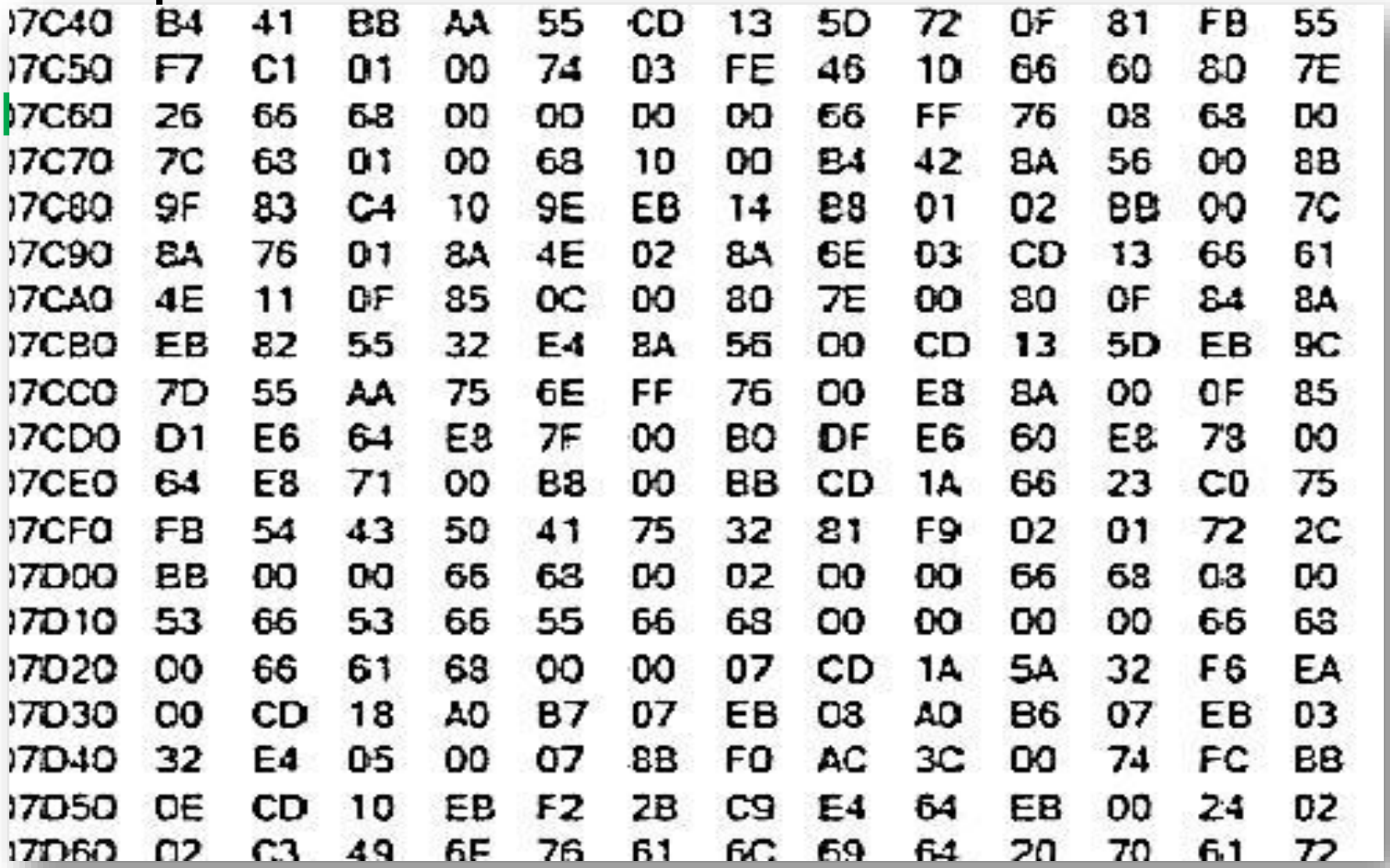


High Level → Micro Code

- High Level language
  - Formulating program for certain application areas
  - **Hardware independent**

- **Assembler** languages
  - **Machine oriented** **language**
  - Programs orient on special hardware properties
  - More comfortable than machine code
    (e.g. by using **symbolic notations**)

High Level → Micro Code

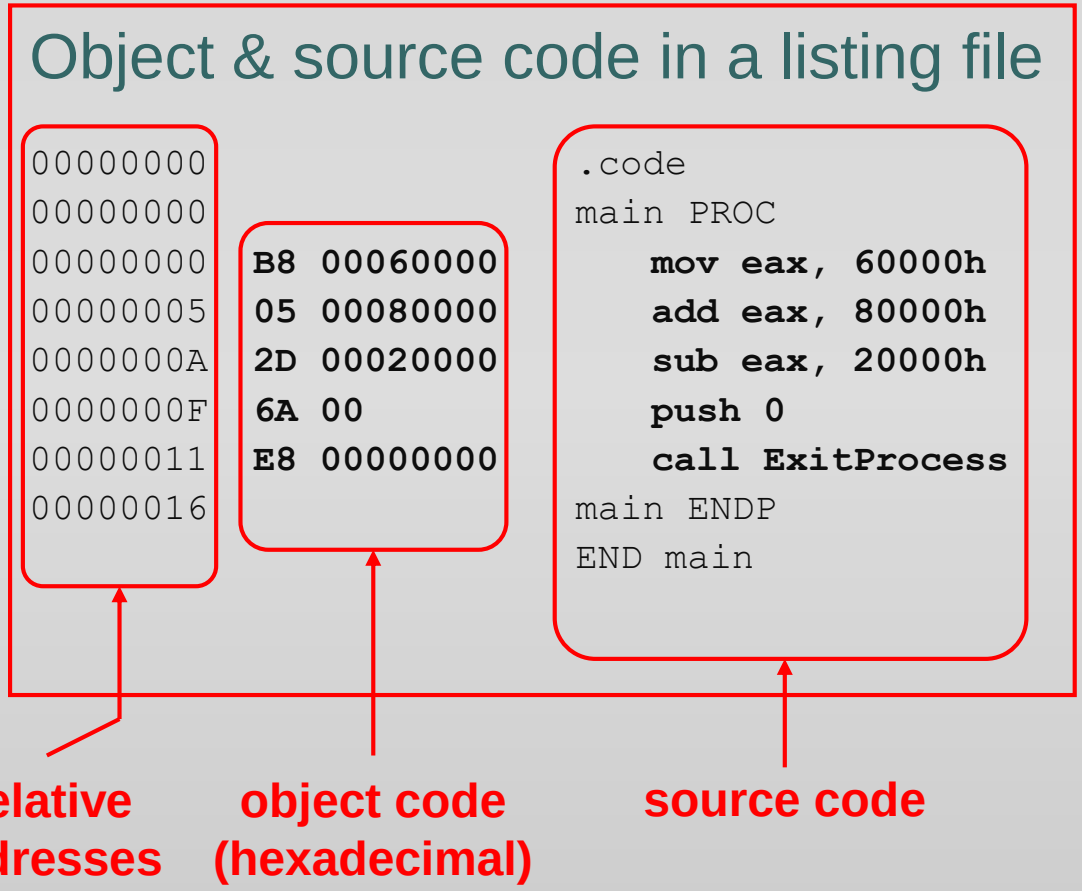


```
07C40  B4 41 BB AA 55 CD 13 5D 72 0F 81 FB 55
07C50  F7 C1 01 00 74 03 FE 46 10 66 60 80 7E
07C60  26 66 68 00 00 00 00 66 FF 76 08 68 00
07C70  7C 63 01 00 68 10 00 B4 42 8A 56 00 8B
07C80  9F 83 C4 10 9E EB 14 B8 01 02 BB 00 7C
07C90  8A 76 01 8A 4E 02 8A 6E 03 CD 13 66 61
07CA0  4E 11 0F 85 0C 00 80 7E 00 80 0F 84 8A
07CB0  EB 82 55 32 E4 8A 56 00 CD 13 5D EB 9C
07CC0  7D 55 AA 75 6E FF 76 00 E8 8A 00 0F 85
07CD0  D1 E6 64 E8 7F 00 B0 DF E6 60 E8 78 00
07CE0  64 E8 71 00 B8 00 BB CD 1A 66 23 C0 75
07CF0  FB 54 43 50 41 75 32 81 F9 02 01 72 2C
07D00  BB 00 00 66 68 00 02 00 00 66 68 08 00
07D10  53 66 53 66 55 66 68 00 00 00 00 66 68
07D20  00 66 61 68 00 00 07 CD 1A 5A 32 F6 EA
07D30  00 CD 18 A0 B7 07 EB 08 A0 B6 07 EB 03
07D40  32 E4 05 00 07 8B F0 AC 3C 00 74 FC BB
07D50  0E CD 10 EB F2 2B C9 E4 64 EB 00 24 02
07D60  02 C3 49 6E 76 61 6C 69 64 20 70 61 72
```

Listing File

Contains

- Source code
- Object code
- Relative addresses
- Segment names
- Symbols
  - Variables
  - Procedures
  - Constants



Assembler languages

- Translated into machine code language
- Each operation code owns one symbolic command

For example (for **x86** processor), the instruction below tells to **move an immediate 8-bit value into a register.** The binary code for this instruction is
10110 followed by a 3-bit identifier for which register to use.
The identifier for the **AL** register is **000**, so the following machine code loads the *AL* register with the data **01100001**:

10110000 01100001

This binary computer code has more human-readable hexadecimal form:

B0 61

Here, **B0** means 'Move a copy of the following value into AL', and 61 is a hex representation of the value 01100001, (97_{10}).
Intel assembly language provides the mnemonic **MOV** for instructions such as this, so the machine code above can be written as follows in assembly language:

```
MOV  AL, 61h   ; load AL with 97 dec (61 hex)
```

This is much easier to read and to remember.

- Labels for command addresses

Assembler - Structure

Label	Mnemomic	Operand	Comments

- **Label** symbolic labeling of an assembler address (command address at Machine level)
- **Mnemomic -** symbolic description of an operation
- **Operands -** Contains of variables or addresse (if necessary)
- **Comments**

total: **MOV AL, 61h** **;** load AL with 97 dec (61 hex)

Assembler - Machine Instructions

Bitpatterns are created, executed as commands by CPU

- Classes:
  - **Arithmetic/logical Operations(ADD,SUB,XOR**, administrative commands - **EQU, shifting & rotation**)
  - **Data transfer** (load/save operations,
    RAM ←→ register,
    register ←→ register)
  - **Control commands**(jump op. [un-]conditional /relativ,control op. – STOP)
  - **In-/output** commands

Assembler Instructions (Pseudo Commands)

- Instructions to assembler
  - Controlling translation process
  - No creation of machine code
  - Affect creation of machine instructions
- Types:
  - Program organisation
  - equations and symbolic Addresses
  - Definition of Constants and Memory
  - Addressing

Assembler – All purpose Register

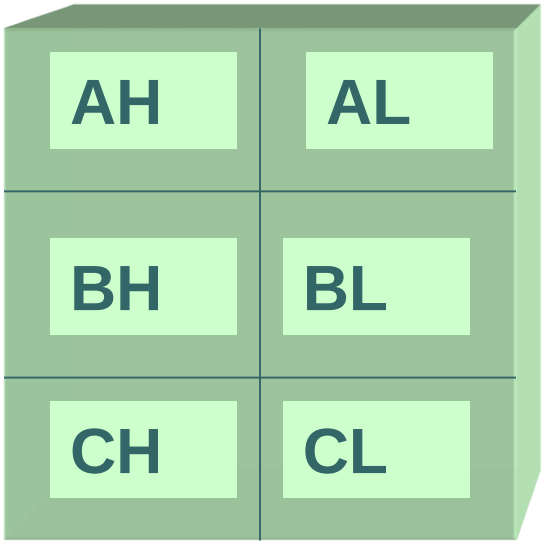
Arithmetic example:

- Source and Destination Data width has to euqal
- AX , BX, CX, DX, SI, DI, BP, SP

; **arithmetic operations**

```
ADD  AX, BX    ; AX := AX+BX
SUB  AH, AL    ; AH := AH - AL
MOV  AL, CL    ; AL := CL
INC  CX        ; CX := CX+1
DEC  CL        ; CL := CL-1
NEG  CX        ; CX := -CX, inna nazwa tej operacji to uzupełnienie dwójkowe
```

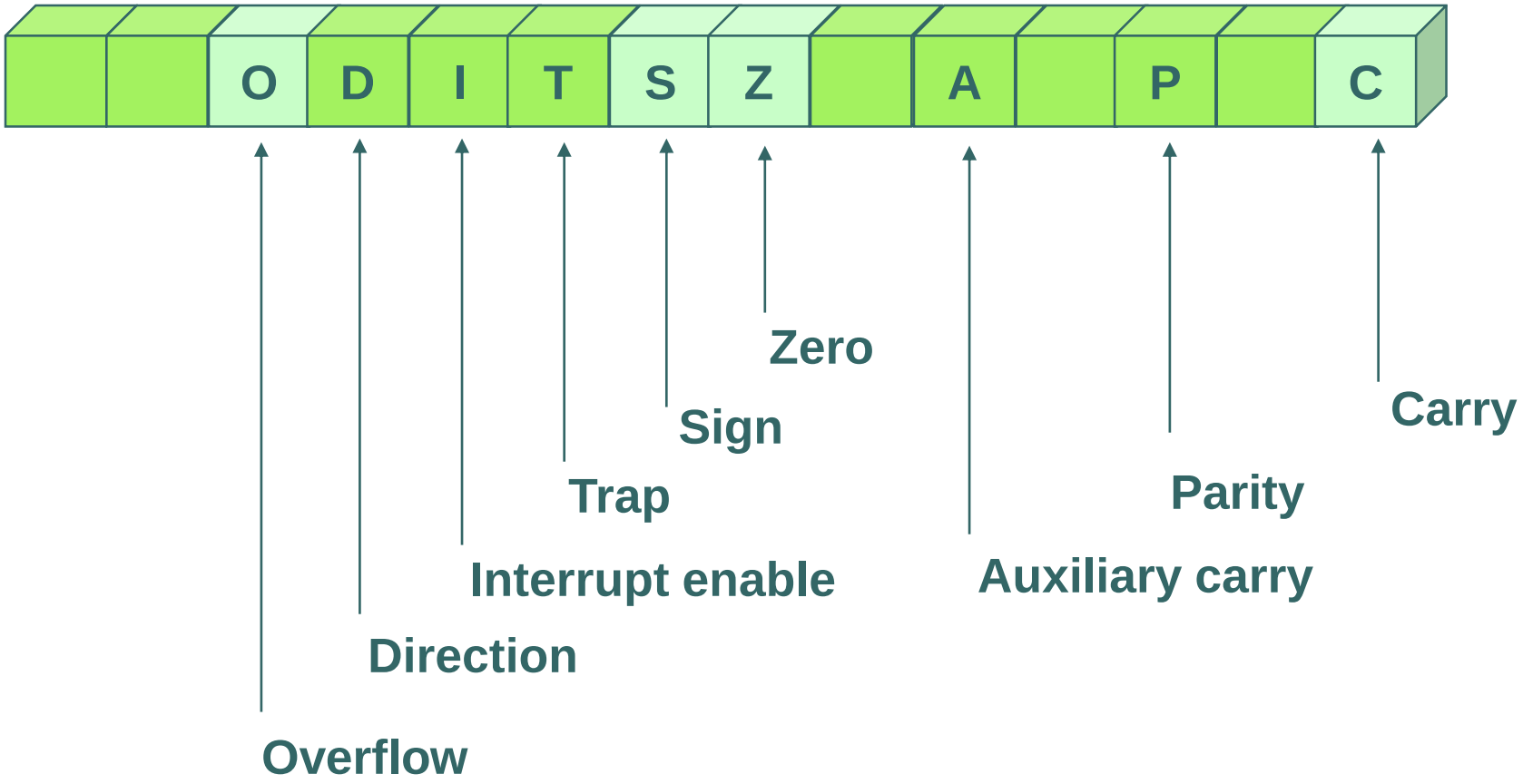
All purpose Register



Assembler – Special Register

- Unless to all-purpose registers
  - Special register(SS, DS, CS, ES, IP)
    - **Never ever are**
      - **Destination/Source of a „mov“ command**
      - **Destination of arithmetic operations**

Assembler – Flag Register



Assembler – Flag Register

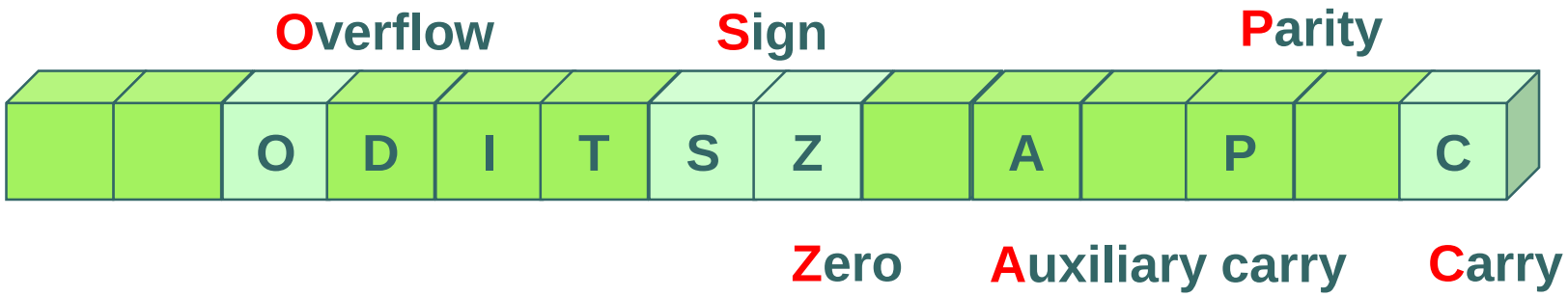
FLAG-Bits:

C Carry	**Area crossing of unsigned numbers**
A Aux. Carry	**Area crossing at BCD-design**
O Overflow	**Area crossing at arithmetic operation with signed numbers**
S Sign	**True if result = negativ**
Z Zero	**Result = Null**
P Parity	**Result has an even number of 1 Bits**
D Direction flag	**Defines direction of string-commands**
I Interrupt	**Global Interrupt Enable/Disable Flag**
T Trap Flag	**Used by debugger, allows single-step-modus**

Assembler – Flag Register

- Operations & flags

ADD, SUB, NEG	**affects**	**O, S, Z, A, P, C**
INC, DEC	**-“-**	**O, S, Z, A, P**
MUL, DIV	**-“-**	**O, C**
AND, OR , XOR	**-“-**	**S, Z, P, C**



Assembler – Jump Operations

Un-/conditioned jumps (Example)

```
        MOV  CX,7h
        MOV  AX, 0
        CMP  CX, 0
again:  JZ   end      (jumpzero, conditioned j.)
        ADD  AX, CX
        DEC  CX
        JMP again     (unconditioned jumped)
end:    NOP
```

Example listing of assembly language source code

```
; zstr_count:       counts a zero-terminated ASCII string to determine its size
; in:               eax = start address of the zero terminated string
; out:              ecx = count = the length of the string

                                zstr_count:                 ; Entry point
00000030 B9FFFFFFFF             mov ecx, -1                 ; Init the loop counter, pre-decrement
00000035 41             .loop:  inc ecx                     ; Add 1 to the loop counter
00000036 803C0800               cmp byte [eax + ecx], 0     ; Compare the value at the string's [starting
                                                              memory address + the loop offset], to zero
0000003A 75F9                   jne .loop                   ; If the memory value is not zero,  then jump
                                                            ; to the label '.loop', elase continue next line
                        .done:                  ;  We don't do a final increment, because even  though the count
                                                ; is base 1, we do not include the zero terminator in the string's length
0000003C C3                       ret                       ; Return to the calling program
```

The 1st column is the relative address, in hex, of where the code will be placed in memory.
The 2nd column is the actual compiled code. For instance, B9 is the x86 opcode for the MOV ECX instruction; FFFFFFFF is the value −1 in two's-complement binary form.
Names suffixed with colons (:) are symbolic labels; the labels do not create code, they are simply a way to tell the assembler that those locations have symbolic names.
Prefixing a period (.) on a label is a feature of the assembler, declaring the label as being local to the subroutine.