

# Assembler, HUB

ISU cv 2

[www.fit.vutbr.cz/~isakin/isu](http://www.fit.vutbr.cz/~isakin/isu)

# Proč assembler?

- Programování OS
- Překladače
- Optimalizace
- Ladění bez zdrojového kódu

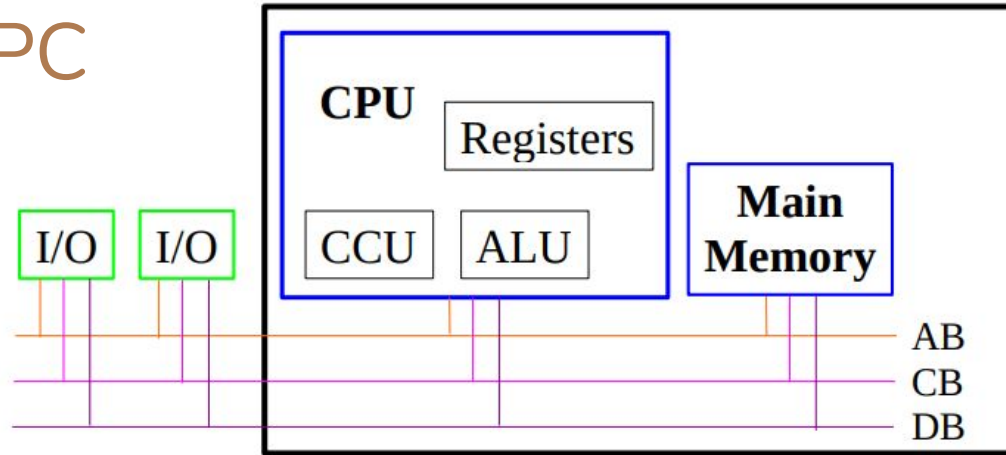
# Co je to assembler?

- Nízkoúrovňový programovací jazyk
- Symbolické instrukce procesoru
- Překlad jazyka na jiný jazyk
- Po překladu binárka (0,1)
- Pracuje přímo s registry



Hardware

# Schéma PC



- |            |                                    |
|------------|------------------------------------|
| CPU        | Central Processing Unit (procesor) |
| ALU        | Arithmetic and Logic Unit          |
| CCU        | Central Control Unit (řadič)       |
| I/O        | Input/Output Unit                  |
| AB, CB, DB | Address Bus, Control Bus, Data Bus |

# Pojmy a průjmy

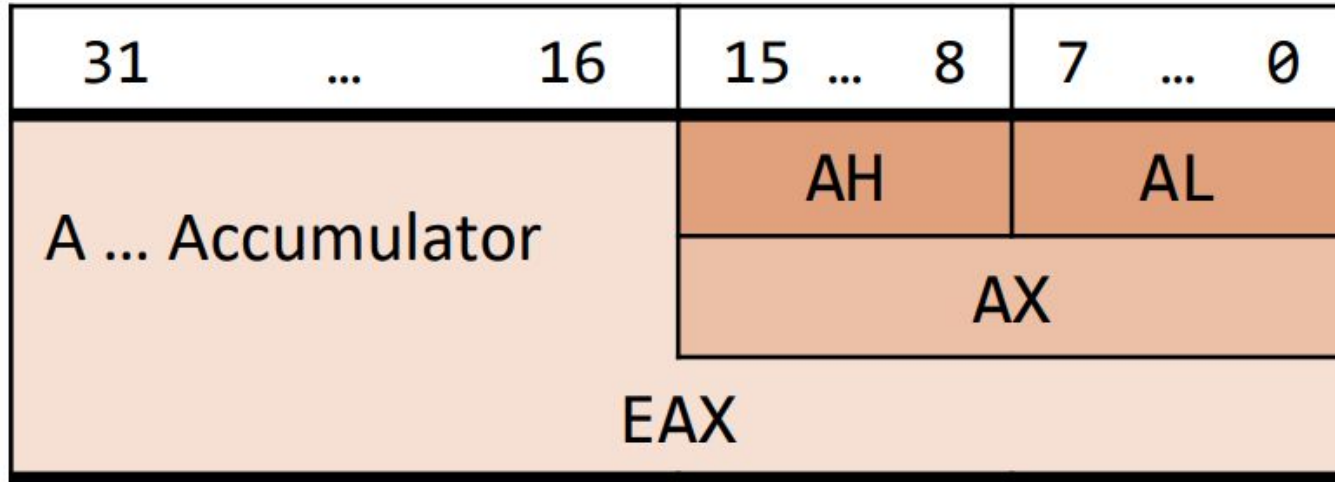
- **Procesor**

- CPU = Centrální procesorová jednotka
- ALU = Aritmeticko-logická jednotka
  - Procesor vždy pracuje s celými čísly v doplňkovém kódu
- FPU = Floating-point unit = Matematický koprocesor
- budeme pracovat s procesory z rodiny x86 v **32b** režimu

# Pojmy a průjmy

- **Register**
  - velmi malé a rychlé úložiště
  - Součástí procesoru, velikost záleží na architektuře
  - Uchovávají data a adresu
- Typy registrů
  - **Datové** ("a", "b", "c", "d") ("a"= EAX || AX || AH || AL)
  - **Ukazatelé** (EIP - ukazuje na následující instrukci, ESP - offset programového zásobníku (push,pop), EBP - referencování proměnných předávaných do subrutiny)
  - **Indexové** (ESI - zdrojový index pro operaci s řetězci, EDI - cílový - || -)

# Register





| 31 ... 16         | 15 ... 8 | 7 ... 0 |
|-------------------|----------|---------|
| A ... Accumulator | AH       | AL      |
|                   | AX       |         |
|                   | EAX      |         |
| B ... Base        | BH       | BL      |
|                   | BX       |         |
|                   | EBX      |         |
| C ... Counter     | CH       | CL      |
|                   | CX       |         |
|                   | ECX      |         |
| D ... Data        | DH       | DL      |
|                   | DX       |         |
|                   | EDX      |         |
| Code Segment      | CS       |         |
| Data Segment      | DS       |         |
| Extra Data Seg.   | ES       |         |
| Stack Segment     | SS       |         |

| 31 ... 16                  | 15 ... 0 |
|----------------------------|----------|
| SP ... Stack Pointer       | SP       |
| ESP                        |          |
| BP ... Base Pointer        | BP       |
| EBP                        |          |
| SI ... Source Index        | SI       |
| ESI                        |          |
| DI ... Destination Index   | DI       |
| EDI                        |          |
| IP ... Instruction Pointer | IP       |
| EIP                        |          |
| registr příznaků           | FLAGS    |
| EFLAGS                     |          |

**H ... High byte, L ... Low byte**

**E ... Extended (= 32bitový režim)**

**další segmentové registry – FS, GS**

# Pojmy a průjmy

- Status register - (E)**FLAGS**

-> příští cvičení

- [https://en.wikipedia.org/wiki/FLAGS\\_register](https://en.wikipedia.org/wiki/FLAGS_register)
- CF - Carry flag - příznak přenosu ven
- PF - Parity flag - sudý počet jedniček
- ZF - Zero flag - výsledek je nula
- SF - Sign flag - záporné číslo
- OF - Overflow flag - znaménkové přetečení

|    |    |      |    |    |    |    |    |    |    |   |    |   |    |   |    |
|----|----|------|----|----|----|----|----|----|----|---|----|---|----|---|----|
| 15 | 14 | 13   | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5 | 4  | 3 | 2  | 1 | 0  |
| 0  | NT | IOPL |    | OF | DF | IF | TF | SF | ZF | 0 | AF | 0 | PF | 1 | CF |

# Pojmy a průjmy

- **Instrukce**

- Označení kódového příkazu pro provedení elementární operace procesoru
- Elementární příkazy určující činnost procesoru
- Jméno (+ operandy), na pořadí záleží!
- název   operand1, operand2  
      ADD     EAX, 5  
      MOV     CX, AX

- Posloupnost instrukcí ⇒ Strojový kód



# Vývojová prostředí

# Vývojová prostředí

- Nasm a GoLink (viz. moodle ISU)
- SASM IDE
- **ISU HUB**
- další

# NASM + GCC

- Terminálový nástroj, který umožňuje překlad kódu v jazyce symbolických instrukcí a jeho ladění.
- Není třeba nic nastavovat (pouze nahrát knihovnu).
- K dispozici na merlinovi

# SASM

- Grafický nástroj, který umožňuje překlad kódu v jazyce symbolických instrukcí a jeho ladění.
- Není třeba nic nastavovat (pouze nahrát knihovnu).
- Kromě funkcí, které budeme potřebovat v tomto prostředí nic jiného nenajdete.

# SASM

The image shows two screenshots of the SASM IDE. The left screenshot shows the source code editor with assembly code. The right screenshot shows the debugger interface with various panels like Memory, Registers, and Output. Callouts in different colors point to specific features and code elements.

**Spuštění** (Start)

**Překlad** (Compile)

**Debugger**

**Velikost vypisované hodnoty (byte, word, ...)** (Size of the value being displayed (byte, word, ...))

**Skok na další breakpoint** (Jump to the next breakpoint)

**Krokování - po řádcích, skok do funkce** (Step - by lines, jump to function)

**Stop**

**Výpis paměti** (Memory dump)

**Zobrazení registrů** (Show registers)

**Změna formátu (bin, hex, ...)** (Change format (bin, hex, ...))

**Pokud pracujete s polem** (If you are working with an array)

```
1  %include "rw32-2018-sasm.inc"
2
3  section .data
4      ; zde budou vase data
5      var db 5
6      msg db 'Hello',0
7      len equ $ - msg
8      count equ 'a'
9  section .bss
10     var1 resb 1
11
12 section .text
13 main:
14     mov ebp, esp ; kvuli debuggovaní
15
16     ; zde bude vas kod
17     mov eax, dword 111
18     mov dl, [msg+1]
19     mov al, byte 5
20     mov bl, len
21     mov [var1], byte 5
22
23     xor eax, eax ; navratovy kod
24     ret ; navrat z funkce
```

[13:07:25] Built successfully.  
[13:07:25] Debugging started...  
[13:26:07] Debugging finished.

[13:35:36] Built successfully.  
[13:35:36] Debugging started...

(Debug mode)

Výpis hodnoty proměnné var: **var**  
Výpis hodnoty z registru ax: **\$ax**  
Výpis adresy len: **&len** (např. při použití *equ*)



# ASM - Knihovna “rw32-2022.inc”

- Předdefinované funkce
- Pro zjednodušení života
- Vytvořené FOgem
- Ke stažení na stránce ISU <https://moodle.vut.cz/course/>



Let's code

# ASM - 3 sekce

- **section .data**
  - Deklarace a definice inicializovaných dat a konstant
  - Globální proměnné se známou počáteční hodnotou
- **section .bss**
  - Deklarace proměnných (definice v programu)
  - Globální proměnné s neznámou počáteční hodnotou
  - Část paměti vyplněná nulami na začátku
- **section .text**
  - Kód, má fixní velikost

# Základní struktura

```
%include 'rw32.inc'
```

```
section .text
```

```
main:
```

```
    ret
```

# Instrukce MOV

```
mov eax, 5           ; vlož 5 (desítkově)
mov ebx, 1010b       ; vlož 10 (binárně)
mov ecx, 0xFF        ; vlož 255 (hexa)
mov edx, ecx         ; přesun mezi registry ECX -> EDX
mov ah, al           ; přesun mezi registry AL -> AH
mov ah, ax           ; přesun mezi registry AX -> AH
mov eax, 1
mov bh, al
```

# Instrukce ADD, SUB

add eax, 5 ; co je v eax?

add eax, ebx

add ah, 1 ; co je v eax?

sub eax, ebx

sub eax, 10b

# Instrukce INC, DEC

```
add eax, 5
```

```
inc eax
```

```
inc al
```

```
dec eax
```

```
dec al
```

# Instrukce CALL

```
%include 'rw32.inc'
```

```
call ReadInt16
```

```
call WriteInt16
```

```
call WriteNewLine
```



# Instrukce NOP

`nop`

Úkoly ...