Procesor, vývojové nástroje a základy programování

ISU-cv02

Ing. Jakub Husa

Vysoké Učení Technické v Brně, Fakulta informačních technologií
Božetěchova 1/2. 612 66 Brno - Královo Pole

ihusa@fit.vutbr.cz

13. února 2024

Procesor

Procesor (CPU) je centrální součástka počítače:

- Výpočty provádí jeho aritmeticko-logická jednotka (ALU) ovládaná řadičem (CCU).
- Procesor pracuje pouze s celými čísly (bez znaménka nebo v doplňkovém kódu).
- Pro práci s desetinnými čísly je zapotřebí matematický koprocesor (FPU, viz. cv11).

Procesor může pracovat buď v 16b (základním) nebo 32b (chráněném) režimu:

- Na cvičeních vždy pracujeme s procesory z rodiny x86 v 32b režimu.
- Výpočet fyzické adresy v základním 16b režimu (p3s14–17) bývá u zkoušky.
- Na hypotetické počítače z druhé přednášky (p2s7–34) můžete zapomenout.

Program procesoru píšeme v jazyce symbolických adres:

- Hodnoty neukládáme do proměnných, ale do registrů.
- Hodnoty nezpracováváme pomocí výrazů, ale instrukcí.

Registry

Registr je malá, rychlá paměťová jednotka umístěná uvnitř těla procesoru:

- Procesor obsahuje celkem deset 32b registrů které nás zajímají.
- Každý registr má svoje jméno a konkrétní účel.
- Názvy registrů jsou case-insensitive – můžeme je psát malými i velkými písmeny.

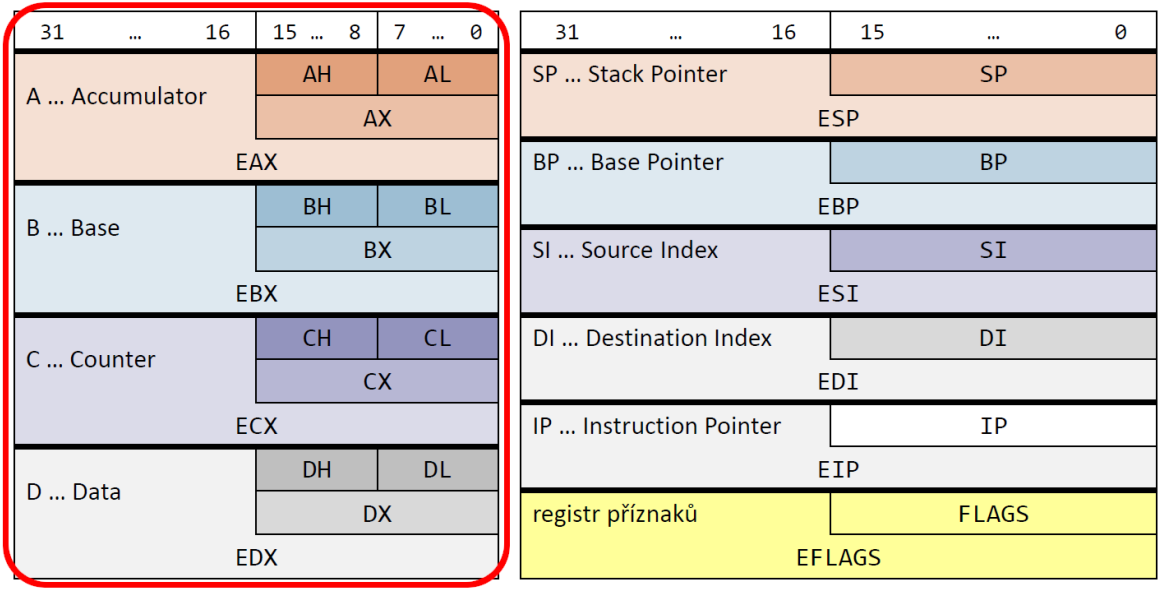
Pro matematické výpočty můžeme používat čtyři obecné 32b registry:

- Accumulator (EAX), Base (EBX), Counter (ECX) a Data (EDX).
- Pokud chceme pracovat s menšími něž 32b čísly, místo celých registrů můžeme používat jejich spodní poloviny (16b) nebo dvě spodní čtvrtiny (8b).
- Spodní poloviny (16b) se nazývají AX, BX, CX a DX.
- Spodní čtvrtiny (8b) se nazývají AH, BH, CH, DH a AL, BL, CL, DL.

Pozor EAX, AX, AH, AL **NEJSOU** čtyři nezávislé registry:

- Například, pokud změníme hodnotu uloženou v AL, změníme tím i hodnoty které uvidíme v AX a EAX (a hodnota v AH se nezmění).

Registry procesoru - obecné registry



Jak bude vypadat obsah uvedených registrů ve dvojkové a desítkové soustavě?

```
                ;|      AX =    0     |
mov   ax,    0  ;|00000000|00000000|
                ;|AH =    0|AL =    0|
                ;|------------------|
                ;|      AX =    1     |
mov   ax,    1  ;|00000000|00000001|
                ;|AH =    0|AL =    1|
                ;|------------------|
                ;|      AX =    2     |
mov   ax,    2  ;|00000000|00000010|
                ;|AH =    0|AL =    2|
                ;|------------------|
                ;|      AX =    0     |
mov   al,    0  ;|00000000|00000000|
                ;|AH =    0|AL =    0|
                ;|------------------|
                ;|      AX =  256     |
mov   ah,    1  ;|00000001|00000000|
                ;|AH =    1|AL =    0|
```

```
                ;|      BX = ????    |
mov   bl,    0  ;|????????|00000000|
                ;|BH = ???|BL =    0|
                ;|------------------|
                ;|      BX = ????    |
mov   bl,  0xF  ;|????????|00001111|
                ;|BH = ???|BL =   15|
                ;|------------------|
                ;|      BX = 3855    |
mov   bh,   bl  ;|00001111|00001111|
                ;|BH =   15|BL =   15|
                ;|------------------|
                ;|      BX =   -1    |
mov   bx,   -1  ;|11111111|11111111|
                ;|BH =   -1|BL =   -1|
                ;|------------------|
                ;|      BX = -256    |
mov   bl,    0  ;|11111111|00000000|
                ;|BH =   -1|BL =    0|
```

Instrukce jsou elementární příkazy jazyka procesoru:

- Každá z instrukcí má nějaké jméno, a některé z nich mohou mít také operandy.
- Operandy od sebe oddělujeme čárkou a na jejich pořadí záleží.
- Některé z instrukcí umějí pracovat jen s konkrétními registry (násobení, viz. cv04).
- Většina instrukcí existuje ve třech velikostech (8b, 16b, 32b), a to kterou z nich použít automaticky odvozuje překladač podle velikosti použitých operandů.

Dostupné instrukce jsou definovány instrukční sadou procesoru:

```
1 mov  eax,  10    ;prirazeni     EAX = 10
2 add  eax,  20    ;soucet        EAX = EAX + 20
3 sub  eax,  30    ;rozdil        EAX = EAX - 30
4 neg  eax         ;negace        EAX = -EAX      (dvojkovy doplnek)
5 xchg eax, ebx    ;vymena        swap(EAX,EBX)
```

- Názvy instrukcí jsou case-insensitive – můžeme je psát malými i velkými písmeny.
- V E-learningu je tabulka základních instrukcí které v ISU budeme potřebovat.
- Kompletní instrukční sadu obsahuje manuál.

Počáteční hodnota registrů je nedefinovaná:

```
1 add  eax,  10   ;EAX = ???
```

Hodnoty v jiné než desítkové soustavě označujeme předponou nebo příponou:

```
2 mov  eax, 0xF   ;EAX =  15
3 mov  eax, 10b   ;EAX =   2
```

Instrukce jsou vykonávány v sekvenčním pořadí:

```
4 mov  eax,  10   ;EAX =  10
5 mov  ebx,  20   ;EAX =  10, EBX =  20
6 add  eax, ebx   ;EAX =  30, EBX =  20
```

Operandy vždy musejí mít stejnou velikost:

```
7 mov   ah,  al   ;ok    (AH = AL)
8 mov   ah,  ax   ;CHYBA (registr AH je 8b, registr AX je 16b)
```

Vyzkoušejte si:

- Jaký bude obsah uvedených registrů v desítkové soustavě?

```
1  mov   ax,   0   ; AH =   0, AL =   0, AX =   0
2  add   ah,   1   ; AH = ???, AL = ???, AX = ???
3  add   ax, 128   ; AH = ???, AL = ???, AX = ???
4  add   al, 128   ; AH = ???, AL = ???, AX = ???
5
6  mov   bx,   0   ; BH =   0, BL =   0, BX =   0
7  add   bl,0x0F   ; BH = ???, BL = ???, BX = ???
8  add   bl,0xF0   ; BH = ???, BL = ???, BX = ???
9  add   bx,  bx   ; BH = ???, BL = ???, BX = ???
10
11 mov   cx,   0   ; CH =   0, CL =   0, CX =   0
12 add   ch, 255   ; CH = ???, CL = ???, CX = ???
13 sub   cl, 128   ; CH = ???, CL = ???, CX = ???
14 neg   cx        ; CH = ???, CL = ???, CX = ???
15
16 mov   dx,   0   ; DH =   0, DL =   0, DX =   0
17 sub   dx,   1   ; DH = ???, DL = ???, DX = ???
18 neg   dh        ; DH = ???, DL = ???, DX = ???
19 xchg  dl,  dh   ; DH = ???, DL = ???, DX = ???
```

Vývojové nástroje a základy programování

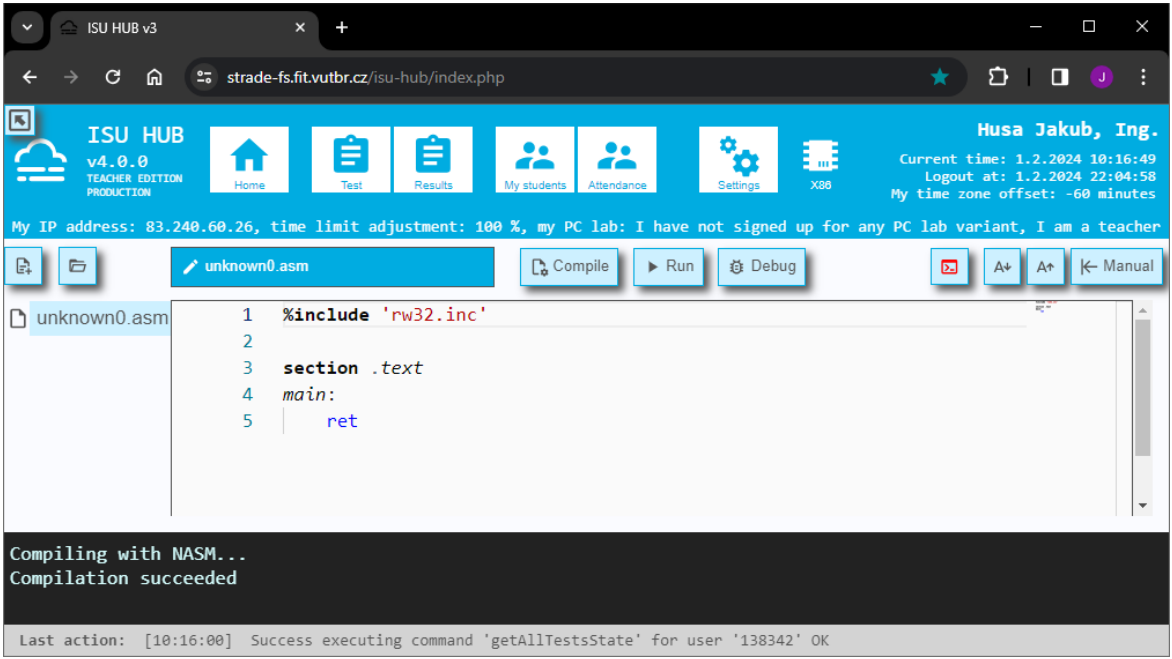
K programování na strojové úrovni budeme potřebovat:

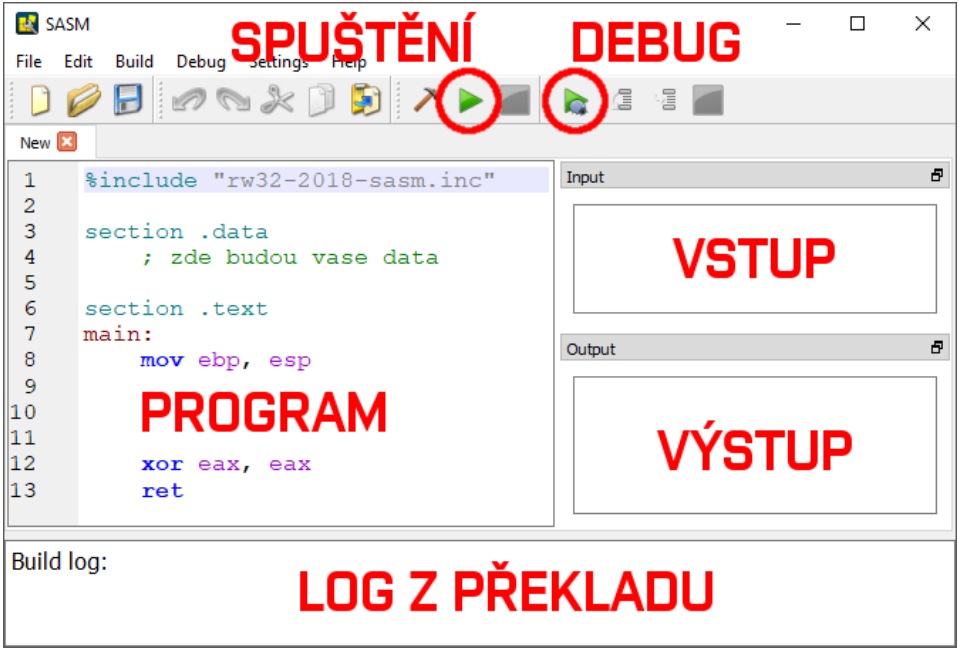
- Editor ve kterém budeme psát program.
- Překladač kterým program přeložíme do strojového kódu.
- Linker kterým strojový kód spojíme s knihovnami na spustitelný soubor.
- Debugger kterým program můžeme krokovat abychom v něm odhalili chyby.

Situaci si výrazně usnadníme použitím nějakého vývojového prostředí:

- Osobně vám doporučuji používat simulátor x86 v ISU-HUBu, ve kterém také budete psát vaše bodované testy.
- Pro off-line programování doporučuji stáhnout používat SASM.
- Další možnosti najdete na E-learningu v osnově 2. cvičení.

Vývojové prostředí ISU-HUB




SASM
SPUŠTĚNÍ
DEBUG
File Edit Build Debug Settings Help
New
1 %include "rw32-2018-sasm.inc"
2
3 section .data
4 ; zde budou vase data
5
6 section .text
7 main:
8 mov ebp, esp
9
10
11
12 xor eax, eax
13 ret
PROGRAM
Input
VSTUP
Output
VÝSTUP
Build log:
LOG Z PŘEKLADU

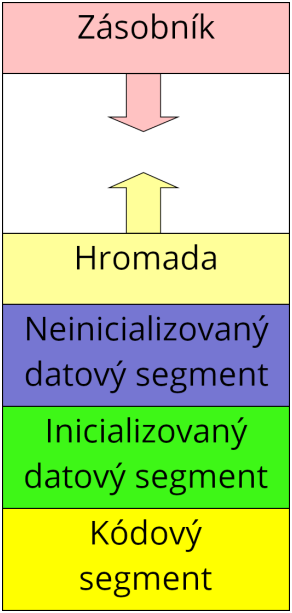
Paměťový prostor programu se dělí na pět segmentů:

- Instrukce píšeme do funkcí, které píšeme do kódového segmentu označovaného jako section .text .
- Kódový segment musí obsahovat hlavní funkci (main) pojmenovanou main: , _main: , nebo CMAIN: , podle zvoleného vývojového prostředí.

Funkce začínají návěštím a končí instrukcí ret (return):

- Návratem z funkce main se ukončí celý program.
- Komentáře označujeme středníkem (;).

```
1 %include 'rw32.inc' ;knihovna pro vstup a vystup
2
3 section .text       ;kodovy segment
4 main:               ;zacatek funkce main
5                     ;telo    funkce main
6     ret             ;konec   funkce main
```



Vstupy a výstupy programu

Nad kódovým segmentem mohou být umístěny datové segmenty (viz. cv03) a nad nimi direktiva %include která umožňuje vložit knihovnu:

- Funkce pro vstup a výstup programu poskytuje knihovna rw32.inc.
- Seznam dostupných funkcí najdete na E-learningu v osnově 7. cvičení.
- SASM používá starší verzi knihovny rw32-2018.inc, kterou najdete ve složce SASM -> Windows -> include -> rw32-2018.inc (manuál je na řádcích 21 až 179).

```
1 %include 'rw32.inc'       ;vlozeni knihovny pro vstup a vystup
```

Funkce voláme instrukci CALL:

- Názvy funkcí jsou case-sensitive – na psaní malých a velkých písmen záleží!
- Knihovní funkce čtou a vypisují pouze z registrů AL (8b), AX (16b), nebo EAX (32b).
- Pro přesun (přiřazení) hodnot mezi registry používáme instrukci MOV.

```
2 call ReadInt32            ;ze vstupu nacti 32b do EAX
3 call WriteInt32           ;na vystup vypis 32b  z EAX
4 call WriteNewLine         ;na vystup vypis konec radku
```

Vyzkoušejte si:

- Program ze vstupu načte dvě 32b čísla (X a Y) a vypíše jejich součet (X+Y).

```
1  %include 'rw32.inc'        ;knihovna pro vstup a vystup
2
3  section .text              ;kodovy segment
4  main:                      ;zacatek funkce main (zacatek programu)
5                             ;                     EAX  EBX
6                             ;                     ???  ???
7      call ReadInt32         ;nacti hodnotu EAX     X   ???
8      mov  ebx, eax          ;EBX = EAX             X    X
9      call ReadInt32         ;nacti hodnotu EAX     Y    X
10     add  eax, ebx          ;EAX = EAX + EBX      Y+X   X
11     call WriteInt32        ;vypis hodnotu EAX    Y+X   X
12
13     ret                    ;konec funkce main (konec programu)
```

Na procvičení

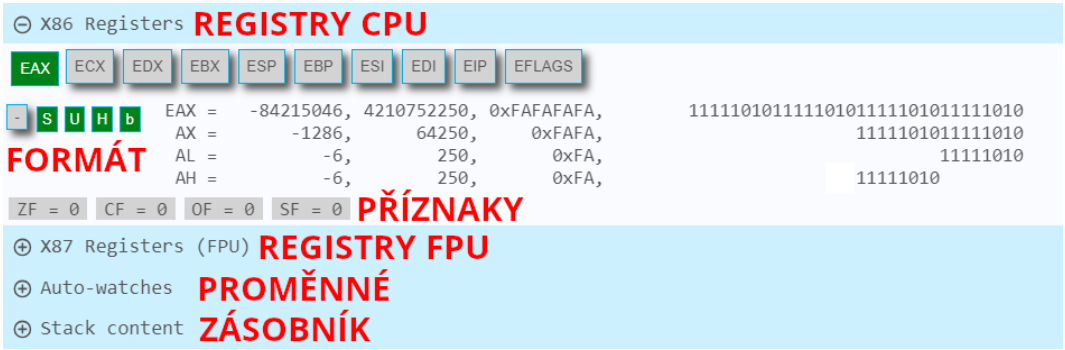
Vyzkoušejte si:

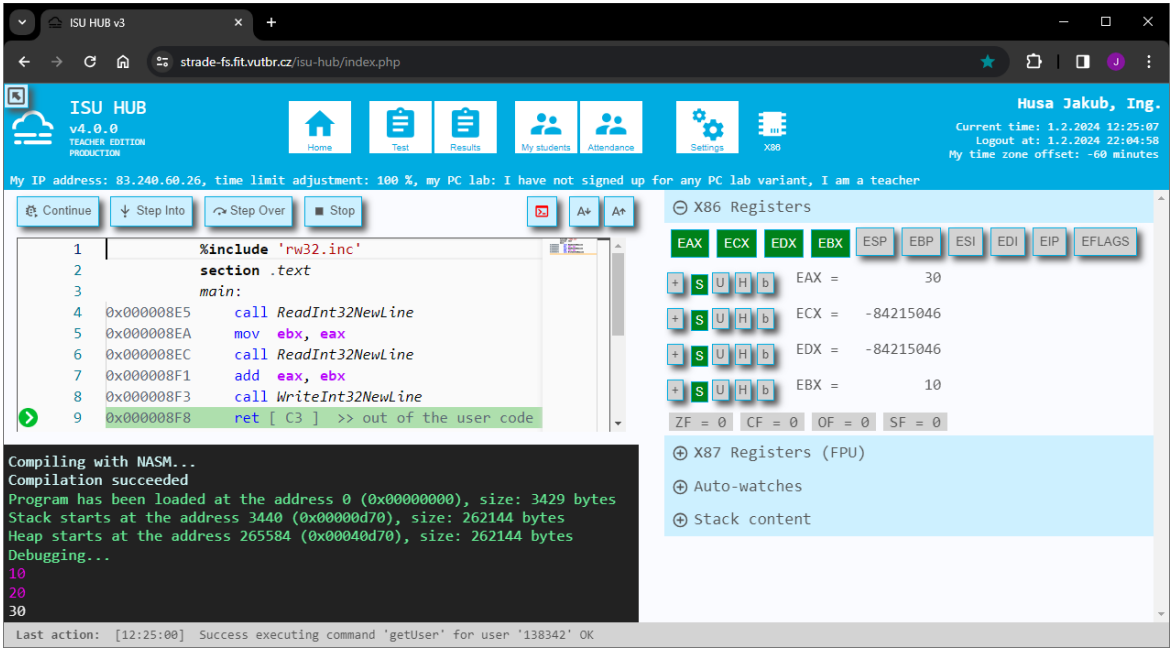
- Ze vstupu načtěte dvě 32b čísla (X a Y), a vypište hodnoty X-Y, -X+Y a -X-Y, každou na vlastní řádek.

K ladění programu používáme debugger:

- Vlevo je zvýrazněná aktuální instrukce, která se provede v následujícím kroku.
- Vpravo se zobrazuje obsah všech registrů CPU, proměnných (viz. cv03), příznaků (viz. cv04), zásobníku (viz. cv08), a registrů FPU (viz. cv11).
- Podobně jako u vyšších programovacích jazyků, program můžeme krokovat po jednotlivých instrukcích, nebo si nastavit breakpoint.

Registry si můžeme vypsat znaménkově, bez-znaménkově, šestnáctkově a binárně:




ISU HUB v3
strade-fs.fit.vutbr.cz/isu-hub/index.php
ISU HUB
v4.0.0
TEACHER EDITION
PRODUCTION
Home
Test
Results
My students
Attendance
Settings
X86
Husa Jakub, Ing.
Current time: 1.2.2024 12:25:07
Logout at: 1.2.2024 22:04:58
My time zone offset: -60 minutes
My IP address: 83.240.60.26, time limit adjustment: 100 %, my PC lab: I have not signed up for any PC lab variant, I am a teacher
Continue
Step Into
Step Over
Stop
1 %include 'rw32.inc'
2 section .text
3 main:
4 0x000008E5 call ReadInt32NewLine
5 0x000008EA mov ebx, eax
6 0x000008EC call ReadInt32NewLine
7 0x000008F1 add eax, ebx
8 0x000008F3 call WriteInt32NewLine
9 0x000008F8 ret [C3] >> out of the user code
X86 Registers
EAX ECX EDX EBX ESP EBP ESI EDI EIP EFLAGS
EAX = 30
ECX = -84215046
EDX = -84215046
EBX = 10
ZF = 0 CF = 0 OF = 0 SF = 0
X87 Registers (FPU)
Auto-watches
Stack content
Compiling with NASM...
Compilation succeeded
Program has been loaded at the address 0 (0x00000000), size: 3429 bytes
Stack starts at the address 3440 (0x00000d70), size: 262144 bytes
Heap starts at the address 265584 (0x00040d70), size: 262144 bytes
Debugging...
10
20
30
Last action: [12:25:00] Success executing command 'getUser' for user '138342' OK

Ladění programu - SASM

Vývojové prostředí SASM také obsahuje debugger:

- Podobně jako u vyšších programovacích jazyků, v programu nejprve nastavíme breakpointy a program krokujeme po jednotlivých instrukcích.
- Levé okno ukazuje zdrojový kód se zvýrazněnou aktuální instrukcí.
- Pravé okno zobrazuje aktuální obsah všech registrů.

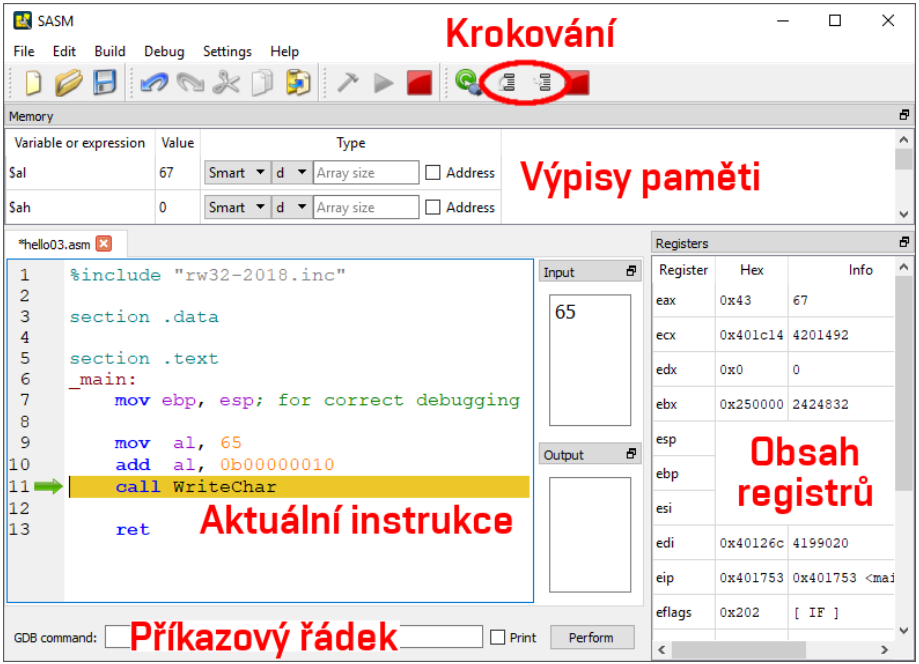
Horní okno slouží k zobrazení obsahu proměnných (viz. cv03):

- Umožňuje ale vypsat i obsah registrů CPU a jejich částí.
- Jména musíme psát malými písmeny a s předponou $ ($ah, $al, $ax, $eax).

Výpisům můžeme nastavit různé formátování:

- Smart (automaticky), Hex (šestnáctková), Bin (dvojková), Char (znak ASCII), Int (desítková), Uint (desítková bez znaménka) a Float (desetinné číslo).

V dolní části je také příkazová řádka GDB (nad rámec ISU).


SASM
File Edit Build Debug Settings Help
Krokování
Memory
Variable or expression
Value
Type
$al
67
Smart
d
Array size
Address
$ah
0
Výpisy paměti
*hello03.asm
1 %include "rw32-2018.inc"
2
3 section .data
4
5 section .text
6 _main:
7 mov ebp, esp; for correct debugging
8
9 mov al, 65
10 add al, 0b00000010
11 call WriteChar
12
13 ret
Aktuální instrukce
Input
65
Output
Registers
Register
Hex
Info
eax 0x43 67
ecx 0x401c14 4201492
edx 0x0 0
ebx 0x250000 2424832
esp
ebp
esi
Obsah registrů
edi 0x40126c 4199020
eip 0x401753 0x401753 <mai
eflags 0x202 [IF]
GDB command:
Příkazový řádek
Print
Perform

Na procvičení

Vyzkoušejte si:

- Ze vstupu načtěte čtyři 8b čísla (W, X, Y, Z), a vypište hodnoty 100-X, 2Y+4Z a W+X+Y+Z, každou na vlastní řádek.