

ASSEMBLER



Univerziteta u Novom Sadu
Fakultet tehničkih nauka
Departman za računarstvo i automatiku
Odsek za primenjene računarske nauke i informatiku
Katedra za informatiku

Dr Željko Marčičević

Vrste programiranja



- **Programski jezici visokog nivoa:**
 - programiranje u **procedurnim** programskim jezicima (C, C++, JAVA,...)
 - zanemaruje arhitekturu računara
- **Programski jezici niskog nivoa:**
 - programiranje **mašinskim i asemblerskim** programskim jezicima
 - zahteva detaljno poznavanje arhitekture računara
 - programiranje je teže
 - manje pregledan kod
 - teže održavanje programa
 - maksimalno iskorišćenje mogućnosti računara

Zašto Asembler?



- Kada je potrebno maksimalno iskoristiti sve resurse računara.
- Današnji prevodioci (kompajleri) su veoma napredovali po pitanju optimizacije, ali i njihove mogućnosti su ograničene.
- Kako bi dobili što optimalniji kod, potrebno je da se spustimo na najniži nivo programiranja – asembler.

SASM okruženje za asemblersko programiranje



- **SASM** (*SimpleASM*) – jednostavna besplatna platforma otvorenog koda (eng. *Open Source*) za asemblerske jezike.
- Omogućava kreiranje jednostavnih asemblerskih programa:
 - uređivanje koda
 - prevođenje i povezivanje
 - testiranje i ispravljanje grešaka
- **Prednosti i nedostaci** SASM okruženja:
 - Jednostavna instalacija (Windows/Linux)
 - Pogodno je za početnike u asemblerskom programiranju
 - Nedostatak: nije prilagođen većim projektima!
 - Link za preuzimanje SASM okruženja: <https://dman95.github.io/SASM/english.html>

Instalacija - Windows



- Preuzeti odgovarajući instalacioni program sa zvanične web stranice.
- Pokrenuti instalaciju i pratiti uputstva na ekranu.
- Windows verzija sadrži sve što je potrebno za rad sa asemblerskim jezikom za Intel 80386

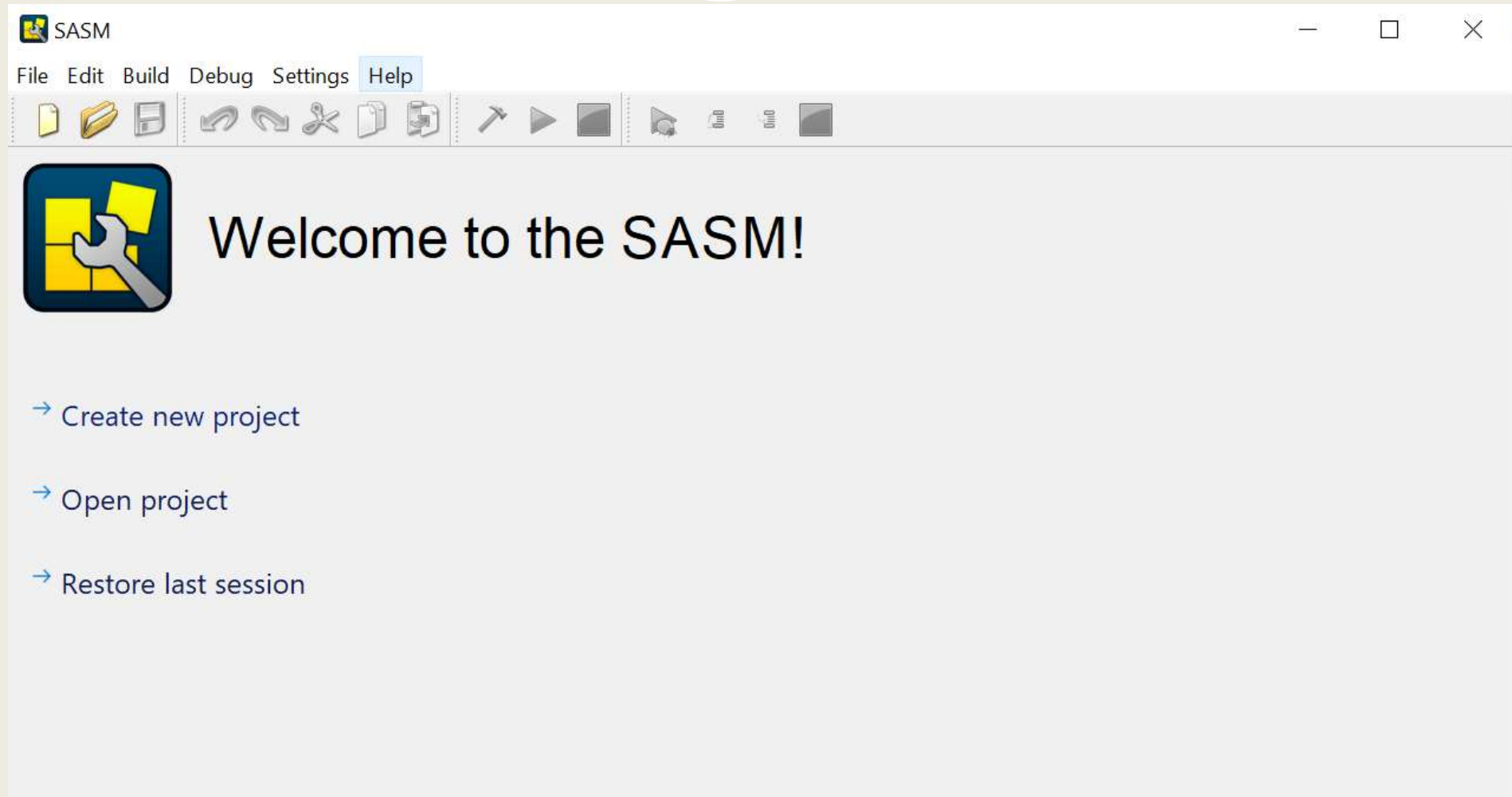
Instalacija - Linux



- Na zvaničnoj web stranici postoje spremni paketi za neke distribucije:
 - *Debian*
 - *Fedora*
 - *openSuse*
 - *xUbuntu*
- Za ostale distribucije potrebno je prevođenje izvornog koda:

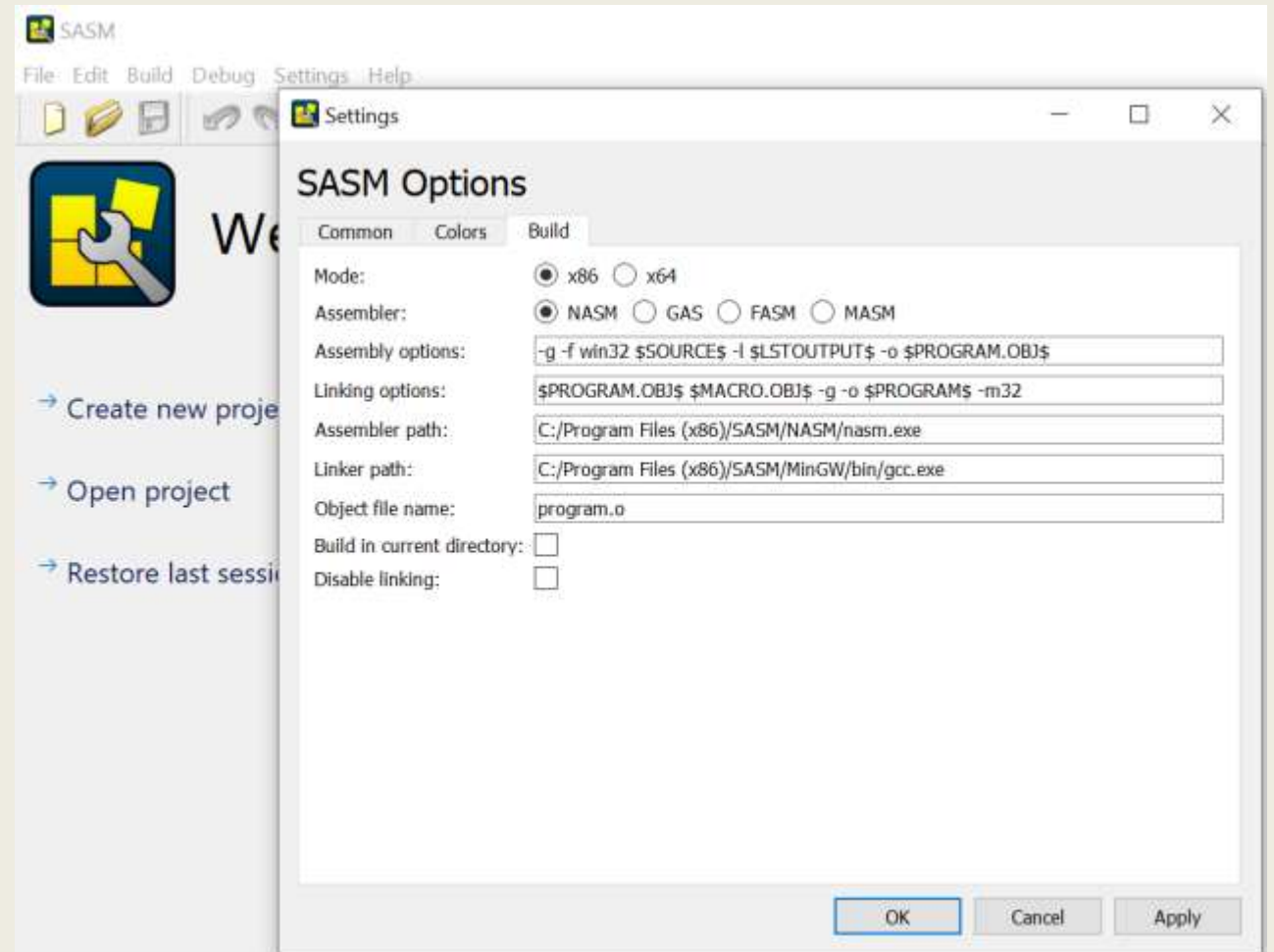
```
$ tar xzvf Dman95-SASM-v3.9.0-0-g308c525.tar.gz  
$ cd Dman95-SASM-308c525/  
$ qmake  
$ make  
$ make install # (pokrenuti kao root)
```
- **NAPOMENA:** Pozadinski programi (eng. *backend*) se posebno instaliraju pre instalacije SASM okruženja (kompajler **gcc**, **nasm**, debugger **gdb** ili **ddd**)

Pokretanje SASM okruženja



Podešavanja

- **Settings → Settings → Build:**
- SASM podržava više asemblera:
 - **NASM**
 - GAS (GNU assembler)
 - FASM
 - MASM
- Podržava i različite arhitekture:
 - **X86** (32-bitna Intel arhitektura)
 - X64 (64-bitna Intel arhitektura)



Napomene i ograničenja

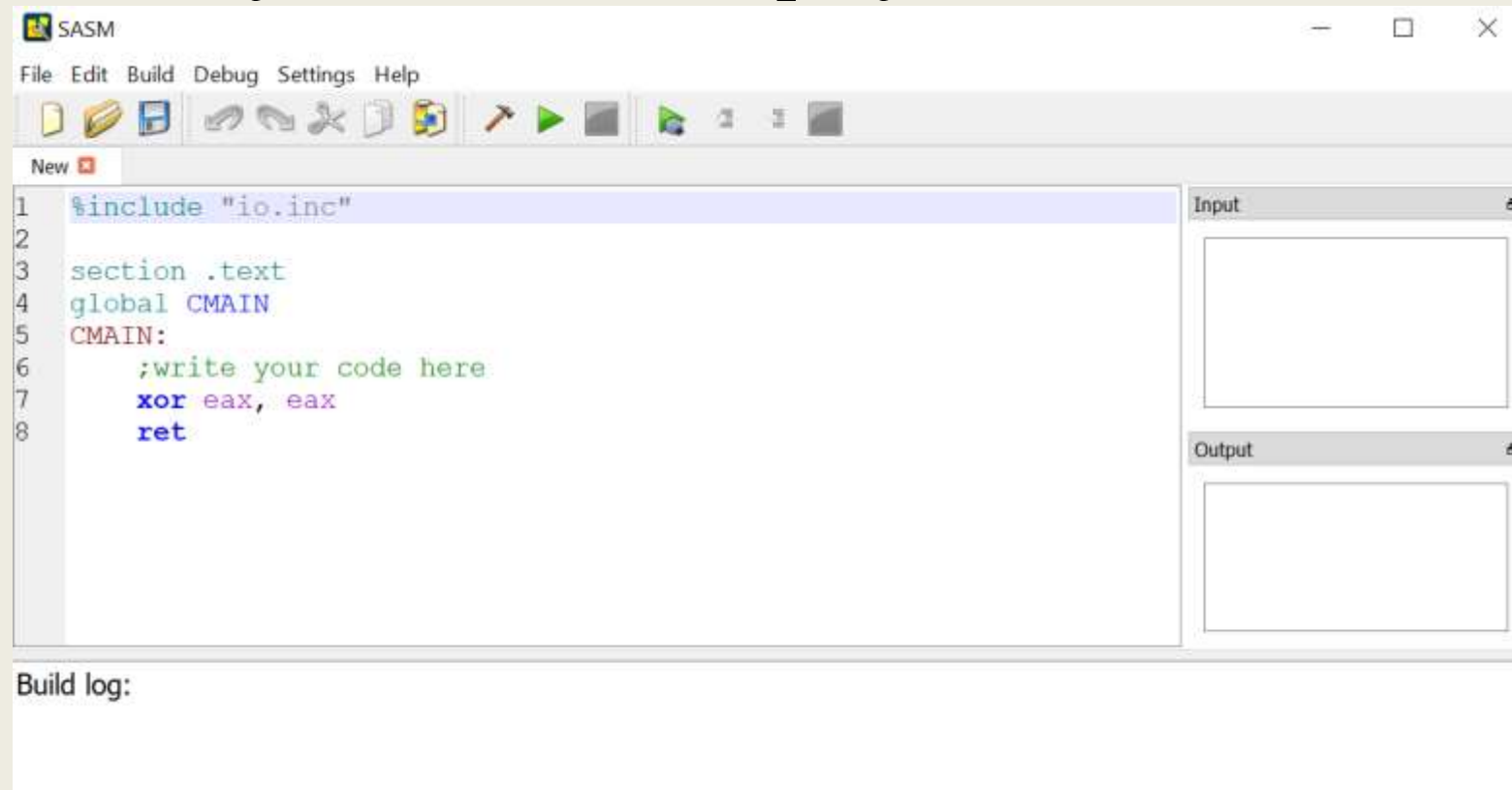


- Ukoliko koristite 32-bitni operativni sistem, ne možete prevoditi i izvršavati 64-bitne programe
- Ukoliko koristite **64-bitni Windows**, tada se uz prethodna podešavanja mogu prevoditi i pokretati 32-bitni programi.
- Ukoliko koristite **64-bitni Linux**, tada je neophodno instalirati multilib pakete (*gcc-multilib*, 32-bitnu verziju standardne C biblioteke, i sl.):
 - Jednostavnija varijanta je korišćenje 32-bitne Linux virtuelne mašine

Kreiranje novog projekta



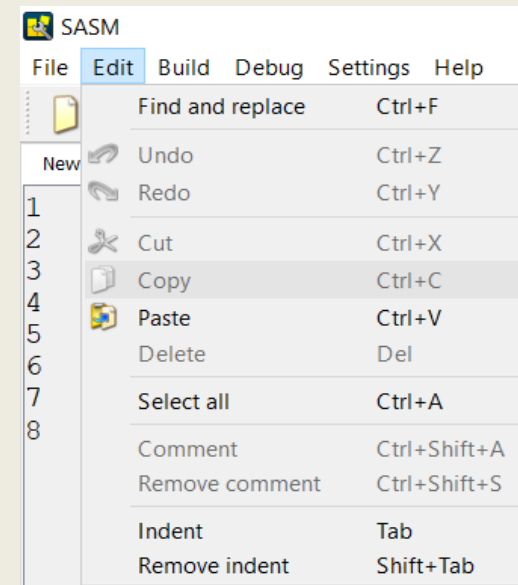
- **Create New Project** sa početnog ekrana **ili** izborom opcije **File→New** iz menija, kreira se novi projekat:



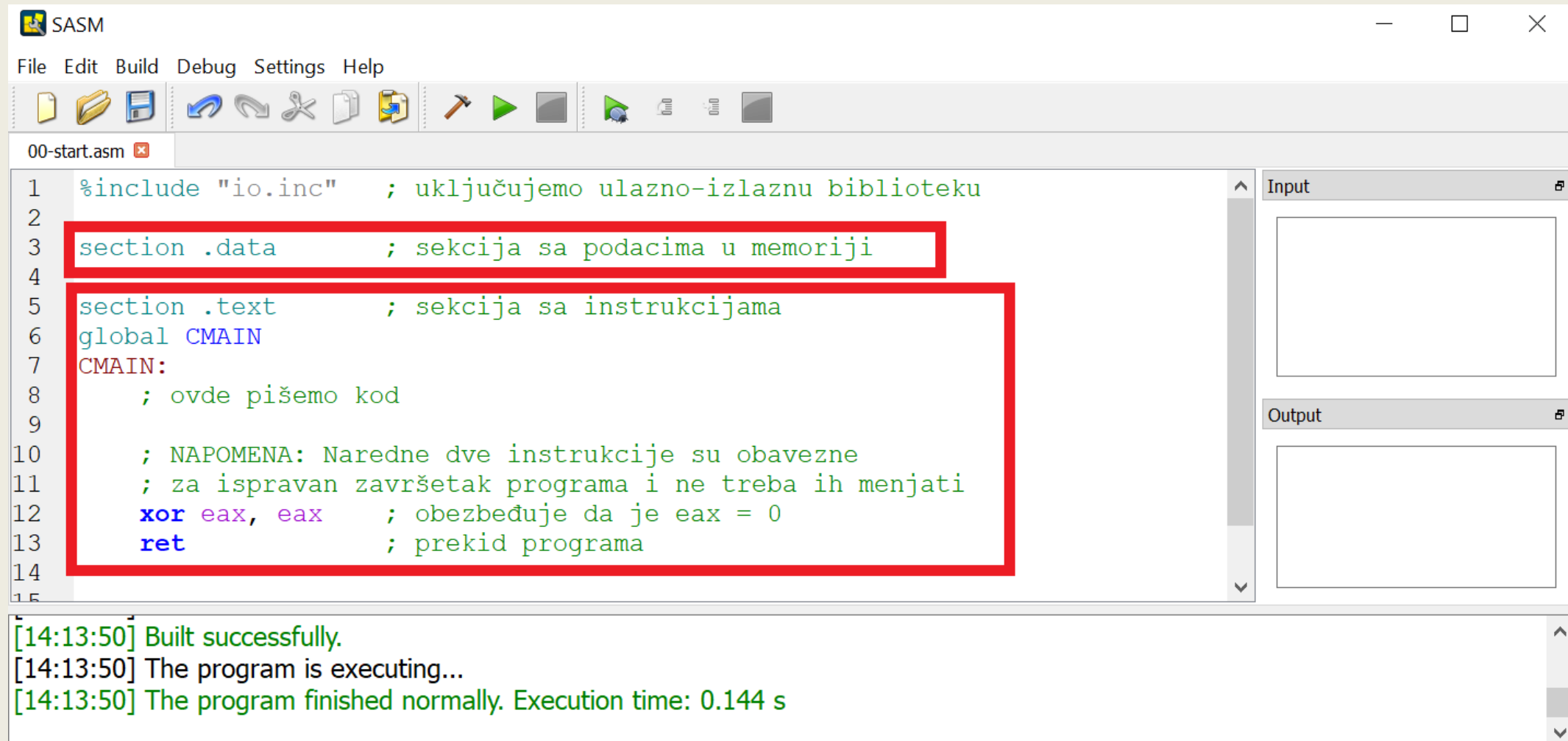
Uređivanje koda



- **1 datoteka = 1 projekat** (nije moguće raditi sa složenim projektima koji sadrže više datoteka)
- Moguće je uređivati više datoteka istovremeno (u više jezičaka)
- Najveći deo prozora – **uređivač teksta** (eng. *Text editor*) za pisanje programa
- Opcije za uređivanje u okviru **Edit** menija:
 - *Copy, Cut, Paste, Undo, Redo*
 - *Comment i Remove comment*
 - ✦ za komentarisanje dela koda (ne prevode se)
 - *Find i Replace* – pretraga i zamena u kodu



Sekcije u asemblerskom programu



The screenshot displays the SASM (Simple Assembler) application window. The title bar reads 'SASM'. The menu bar includes 'File', 'Edit', 'Build', 'Debug', 'Settings', and 'Help'. The toolbar contains icons for file operations (new, open, save, print), editing (undo, redo, cut, copy, paste), and execution (assemble, run, stop, single step, watch, memory dump). The main window shows a file named '00-start.asm' with the following assembly code:

```
1  %include "io.inc"    ; uključujemo ulazno-izlaznu biblioteku
2
3  section .data         ; sekција sa podacima u memoriji
4
5  section .text         ; sekција sa instrukcijama
6  global CMAIN
7  CMAIN:
8      ; ovde pišemo kod
9
10     ; NAPOMENA: Naredne dve instrukcije su obavezne
11     ; za ispravan završetak programa i ne treba ih menjati
12     xor eax, eax      ; obezbeđuje da je eax = 0
13     ret               ; prekid programa
14
15
```

Red rectangular boxes highlight the `section .data` and `section .text` declarations, and the code block within the `section .text` starting from `CMAIN:`. On the right side, there are two empty panels labeled 'Input' and 'Output'. At the bottom, a status bar shows the following messages:

```
[14:13:50] Built successfully.
[14:13:50] The program is executing...
[14:13:50] The program finished normally. Execution time: 0.144 s
```

Rezervisane reči, makroi i funkcije



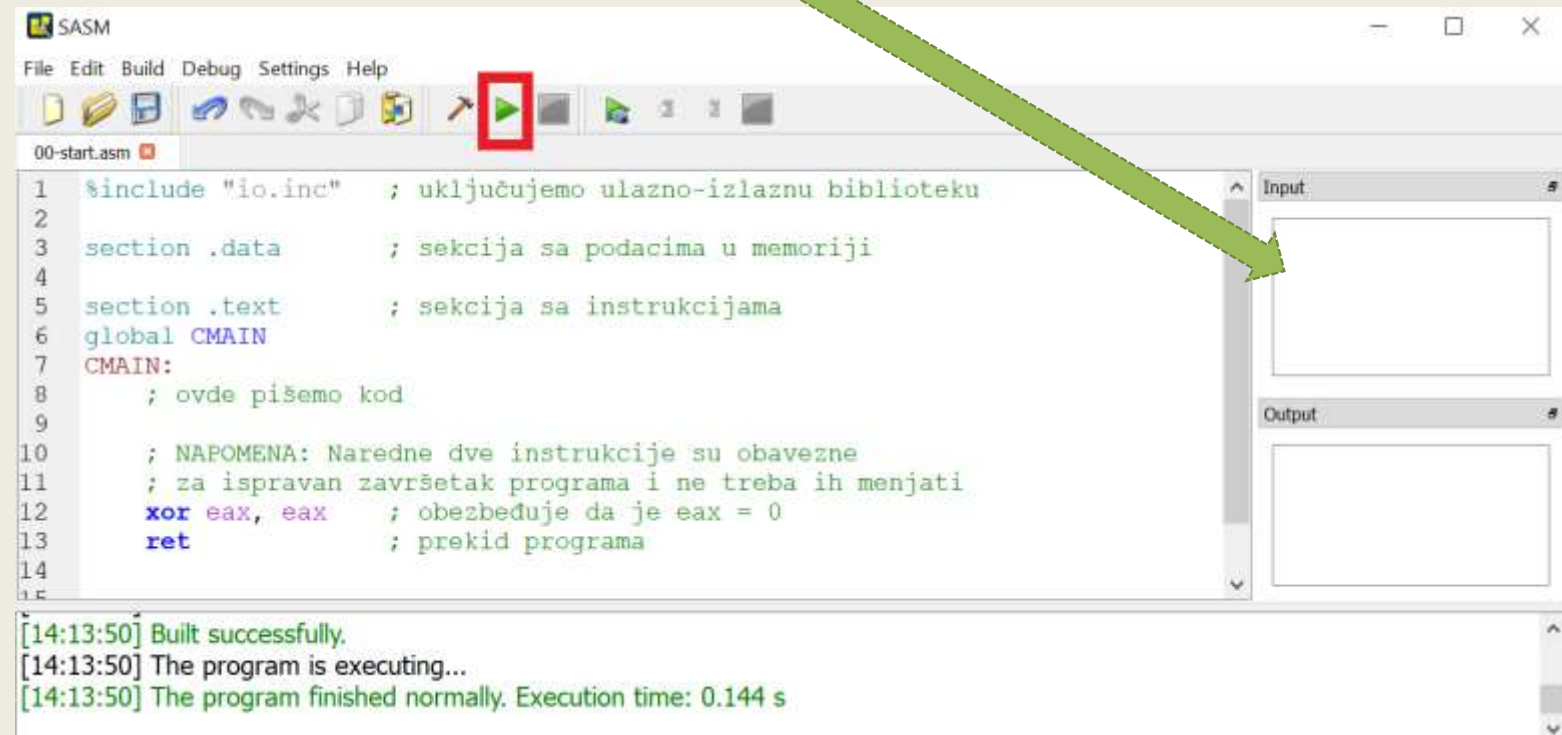
- %include** — uključivanje biblioteka potrebnih za prevođenje i pokretanje programa
- section.data** — sekcija sa podacima (nakon ove direktive se zadaju ulazni i izlazni podaci)
- section.text** — sekcija sa instrukcijama (nakon ove direktive se zadaju asemblerske instrukcije)
- global CMAIN** — deklaracija obavezne funkcije CMAIN (glavna funkcija)
- CMAIN:** — labela za obaveznu glavnu funkciju odakle počinje da se izvršava program
- Svaka **labela** u asemblerskom programu predstavlja **adresu**!
 - Za lakše korišćenje **ulaza i izlaza**, koriste se posebne funkcije (**MAKROI**) i one nisu deo asemblerskih instrukcija!

Naziv <u>makroa</u>	Opis
GET_UDEC size, data GET_DEC size, data	Omogućava unos podataka u decimalnom formatu sa standardnog ulaza (stdin). size - veličina podatka data u bajtima: 1, 2, 4 ili 8. data mora biti konstanta broja ili simbola, ime promenljive, registar ili adresa bez kvalifikatora za veličinu (bajt [], itd.). GET_UDEC posmatra ulazne vrednosti kao neoznačene, a GET_DEC kao označene brojeve.
GET_HEX size, data	Unos i preuzimanje <u>heksadecimalnih</u> podataka sa standardnog ulaza
GET_CHAR data	Unos jednog simbola (<u>karaktera</u>).
GET_STRING data, maxsz	Unos niza karaktera maksimalne dužine maxsz , koji se završava NULOM. maxsz može biti registar ili konstanta
PRINT_UDEC size, data PRINT_DEC size, data	Ispis vrednosti data u decimalnoj predstavi. size - veličina podatka data u bajtima: 1, 2, 4 ili 8. data mora biti konstanta broja ili simbola, ime promenljive, registar ili adresa bez kvalifikatora za veličinu (bajt [], itd.). PRINT_UDEC ispisuje vrednost kao neoznačen, PRINT_DEC - kao označen broj.
PRINT_HEX size, data	Ispisuje vrednosti u heksadecimalnom formatu
PRINT_CHAR ch	Ispisuje simbol ch koji mora biti konstanta broja ili simbola, ime promenljive, registar ili adresa bez kvalifikatora za veličinu (bajt [], itd.).
PRINT_STRING data	Ispisuje tekstualni niz koji se završava NULOM (string).
NEWLINE	Ispisuje novi red „\n“

Prevođenje i pokretanje programa

- Za prevođenje pritisnuti **zeleni trougao** ili **Build**→**Build** iz menija
- Ukoliko program koristi standardni ulaz, **potrebno je pre prevođenja uneti ulazne podatke** u Input prozor.

- **Uspešno preveden program!**



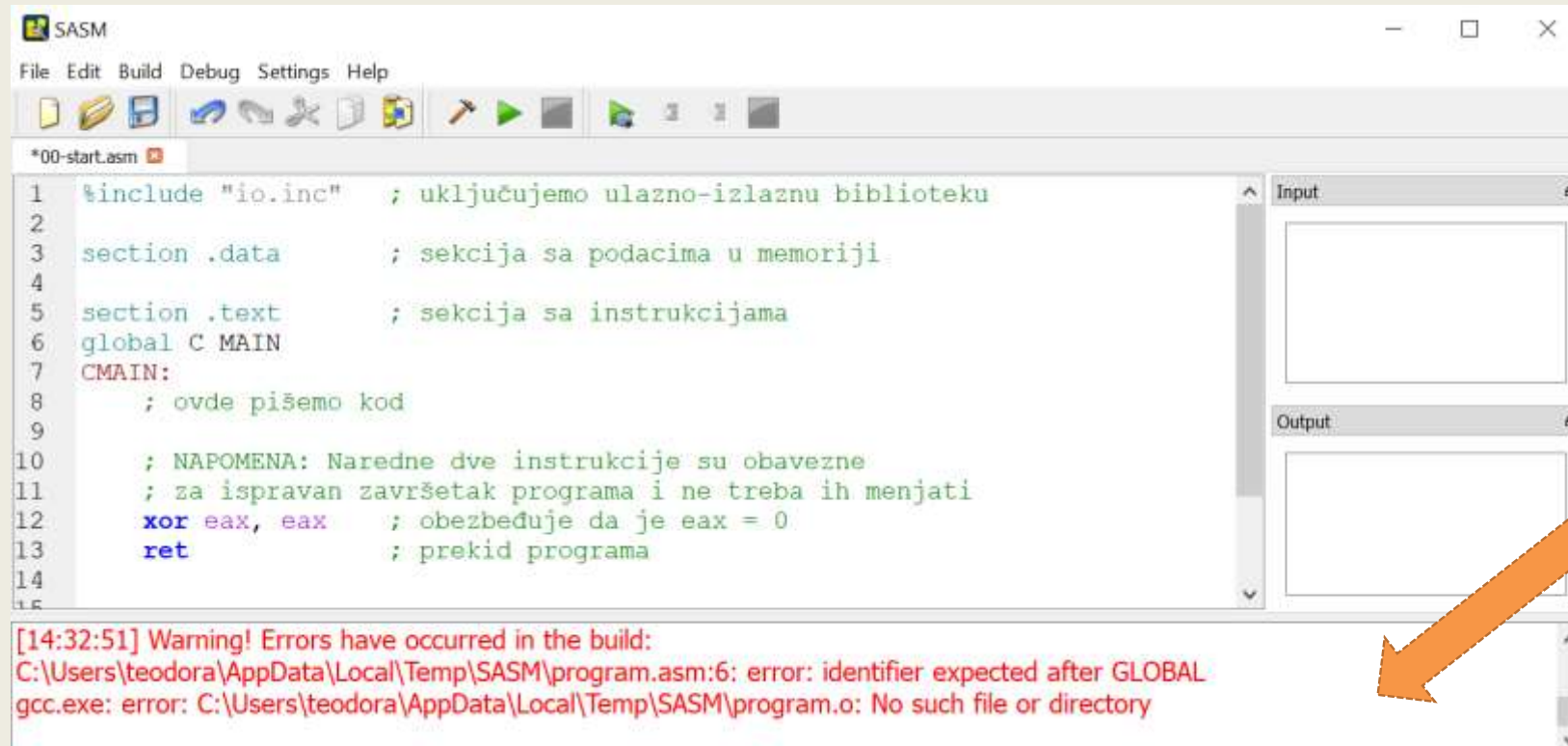
The screenshot shows the SASM IDE interface. The menu bar includes File, Edit, Build, Debug, Settings, and Help. The toolbar contains various icons, with the 'Build' icon (a green triangle) highlighted by a red rectangle. A green arrow points from the text 'zeleni trougao' in the first bullet point to this icon. Another green arrow points from the text 'uneti ulazne podatke u Input prozor' in the second bullet point to the 'Input' text box on the right. The main editor displays assembly code for '00-start.asm' with line numbers 1 through 14. The code includes comments in Serbian and assembly instructions like 'xor eax, eax' and 'ret'. On the right side, there are two empty text boxes labeled 'Input' and 'Output'. At the bottom, a status bar shows the execution logs: '[14:13:50] Built successfully.', '[14:13:50] The program is executing...', and '[14:13:50] The program finished normally. Execution time: 0.144 s'. A green arrow points from the text 'Uspešno preveden program!' to the first log entry.

```
1  %include "io.inc"    ; uključujemo ulazno-izlaznu biblioteku
2
3  section .data        ; sekcija sa podacima u memoriji
4
5  section .text        ; sekcija sa instrukcijama
6  global CMAIN
7  CMAIN:
8      ; ovde pišemo kod
9
10     ; NAPOMENA: Naredne dve instrukcije su obavezne
11     ; za ispravan završetak programa i ne treba ih menjati
12     xor eax, eax      ; obezbeđuje da je eax = 0
13     ret               ; prekid programa
14
```

[14:13:50] Built successfully.
[14:13:50] The program is executing...
[14:13:50] The program finished normally. Execution time: 0.144 s

Greške prilikom prevođenja i pokretanja programa

- Greške se prikazuju u prozoru ispod prozora za editovanje koda i mogu se desiti:
 - Prilikom prevođenja koda** (sintaksne greške)
 - Prilikom izvršavanja programa** (deljenje sa nulom, nedozvoljeni pristup memoriji, itd.)



The screenshot shows the SASM (Small Assembler) interface. The main window displays an assembly file named *00-start.asm. The code includes a header file 'io.inc', defines data and text sections, and contains a main routine. The code is as follows:

```
1  %include "io.inc"    ; uključujemo ulazno-izlaznu biblioteku
2
3  section .data        ; sekcija sa podacima u memoriji
4
5  section .text        ; sekcija sa instrukcijama
6  global C MAIN
7  CMAIN:
8      ; ovde pišemo kod
9
10     ; NAPOMENA: Naredne dve instrukcije su obavezne
11     ; za ispravan završetak programa i ne treba ih menjati
12     xor eax, eax      ; obezbeđuje da je eax = 0
13     ret               ; prekid programa
14
```

At the bottom of the window, a warning message is displayed:

```
[14:32:51] Warning! Errors have occurred in the build:
C:\Users\teodora\AppData\Local\Temp\SASM\program.asm:6: error: identifier expected after GLOBAL
gcc.exe: error: C:\Users\teodora\AppData\Local\Temp\SASM\program.o: No such file or directory
```

An orange arrow points from the warning message towards the right side of the window.

Otkrivanje i ispravljanje grešaka

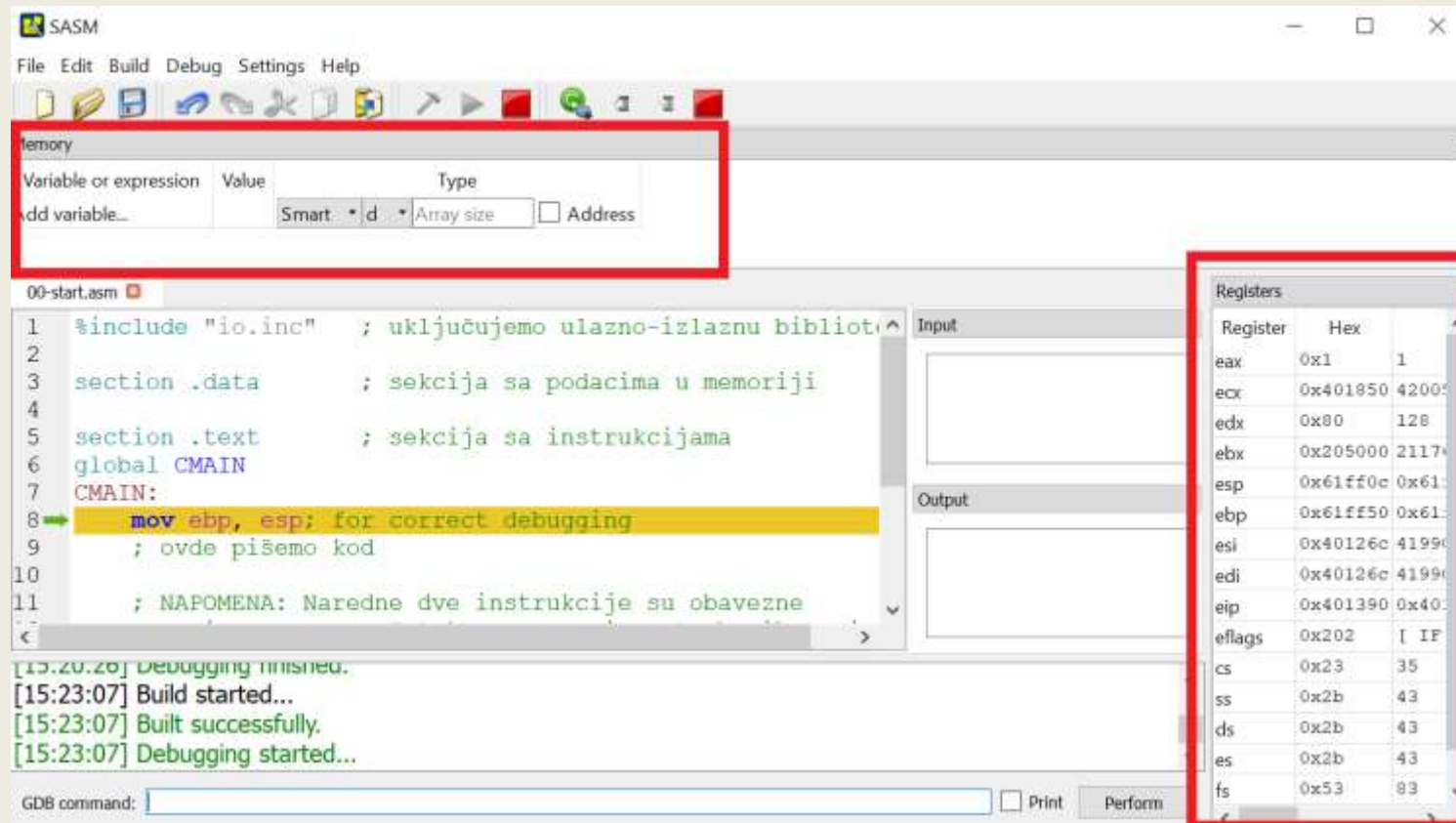


- Za otkrivanje grešaka koristimo alate koji se zovu debageri (eng. *Debugger*).
- Debageri omogućavaju izvršavanje programa (ili dela programa) korak po korak, uz praćenje stanja registara i memorije.
- SASM okruženje koristi alat **GDB** za ovu svrhu (GNU debugger).
- Debugger se pokreće pritiskom **zelenog trougla sa lupom**, ili izborom opcije **Debug**→**Debug** iz glavnog menija.



Praćenje sadržaja registara i memorije

- Nakon pokretanja debagera, moguće je pratiti sadržaj registara i memorije aktiviranjem opcija **Debug**→**Show Registers** i **Debug**→**Show Memory** iz glavnog menija.



Kretanje kroz program



- Debugovanje kreće od prve instrukcije programa i mogu se koristiti sledeće opcije:
 1. **Debug→Step Into** – izvršava tekuću instrukciju i prelazi na sledeću.
 2. **Debug→Step Over** – izvršava tekuću instrukciju, ali u slučaju poziva potprograma ne ulazi u sam potprogram, nego ga u celosti izvršava.
 3. **Debug→Continue** – izvršava ostatak programa odjednom do kraja ili do prve sledeće tačke prekida
 4. **Debug→Stop** – zaustavljanje debugovanja (može i pritiskom na crveni kvadratić u traci sa alatima)

Tačke prekida



- Tačke prekida se koriste za kontrolisano izvršavanje programa i njegovo zaustavljanje na kritičnim mestima, radi debugovanja
- Postavljaju se klikom miša na broj linije u editoru teksta

```
SASM
File Edit Build Debug Settings Help
00-start.asm
1 %include "io.inc" ; uključujemo ulazno-izlaznu biblioteku
2
3 section .data ; sekciya sa podacima u memoriji
4
5 section .text ; sekciya sa instrukcijama
6 global CMAIN
7 CMAIN:
8 mov ebp, esp; for correct debugging
9 ; ovde pišemo kod
10
11 ; NAPOMENA: Naredne dve instrukcije su obavezne
12 ; za ispravan završetak programa i ne treba ih menjati
13 xor eax, eax ; obezbeđuje da je eax = 0
14 ret ; prekid programa
```

[15:23:07] Build started...
[15:23:07] Built successfully.
[15:23:07] Debugging started...
[15:35:58] Debugging finished.

Pomoć



- Opcijom **Help** → **Help** otvara se prozor sa dokumentacijom za SASM okruženje.

Brief help.

SASM (SimpleASM) - simple Open Source crossplatform IDE for NASM, MASM, GAS, FASM assembly languages.

In SASM you can easily develop and execute programs, written in NASM assembly language. Enter code in form and simply run your program. Enter your Input data in "Input" docking field. In "Output" field you can see the result of the execution of the program. Wherein all messages and compilation errors will be shown in the form on the bottom. You can save source or already compiled (exe) code of your program to file and load your programs from file.

SASM supports working with many opened projects - new files are opened and created in new tabs. At the exit from SASM current set of opened files saved. At the next start you can restore previous session. In settings you can set font, color scheme and initial text. SASM is translated into Russian and English. All dialog windows in SASM is docking - you can choose one of many variants of their position.

Standard "Edit" menu extended with abilities to comment/uncomment piece of source code and to create/delete indent with 4 spaces (Tab/Shift+Tab).

Starting with version 2.2 it is possible to reassign the hotkeys. File with them located on the path "Linux/share/sasm/keys.ini" ("usr/share/sasm/keys.ini" if SASM was installed) on Linux and on the path "Windows/keys.ini" ("folder where SASM was installed)/keys.ini") on Windows.

Starting with version 2.3 SASM supports 2 modes - x64 and x86. You can choose mode in settings on "Build" tab. "io.inc" macro library works with both modes. Also there you can change assembler and linker options. For more information about this, see "Building system settings". Starting from version 3.0 you can choose one of four assemblers - NASM, MASM, GAS, FASM in settings on "Build" tab. Also there you can choose your own assembler or linker filling path to them. Thus you can setup SASM on work with any assembler filling path to assembler and, if needed, to linker and filling options for assembling and linking. But debugging and highlighting may work little incorrectly. To realize full support of new assembler, it is needed to implement Assembler abstract class by analogy with already implemented assemblers.

All assemblers (excluding MASM) are included in SASM (on Linux they should be installed) and you can use they right away after their choice. MASM assembler can not be included in the assembly because of its license. To use it, you should install MASM on your computer from site <http://www.masm32.com/> and specify path to MASM assembler (ml.exe, path usually "C:/masm32/bin/ml.exe") and to MASM linker (link.exe, path usually "C:/masm32/bin/link.exe") in according fields on "Build" tab in settings.

SASM contains folder for include files - "Linux/share/sasm/include/" ("usr/share/sasm/include/" if SASM was installed) on Linux and on the path "Windows/include/" ("folder where SASM was installed)/include/" in Windows. But for MASM this folder does not work - In this case you should use absolute path to include files.

"io.inc" macro library for NASM

SASM includes crossplatform input/output library "io.inc". It contains I/O macro and 2 additional macro for NASM: CMAIN - entry point and CEXTERN for invoking functions, located in C language libraries ("CEXTERN printf" for example).

Macro name	Description
PRINT_UDEC size, data	Print number data in decimal representation. size - number, giving size of data in bytes - 1, 2, 4 or 8
PRINT_DEC size, data	(x64). data must be number or symbol constant, name of variable, register or address expression without size qualifier (byte[], etc.). PRINT_UDEC print number as unsigned, PRINT_DEC - as signed.
PRINT_HEX size, data	Similarly previous, but data is printed in hexadecimal representation.
PRINT_CHAR ch	Print symbol ch, ch - number or symbol constant, name of variable, register or address expression without size qualifier (byte[], etc.).
PRINT_STRING data	Print null-terminated text string. data - string constant, name of variable or address-expression without size qualifier (byte[], etc.).
NEWLINE	Print newline ('\n').

Programski model procesora Intel 80386



- 1985. godine
- 32-bitni mikroprocesor
- pripada familiji procesora x86 (*little endian procesor*)
- **CISC** arhitektura (eng. *Complex instruction set computers*):
 - arhitektura sa kompleksnim skupom instrukcija u mnogo različitih formata
 - Različite vrste operanada (neposredni, memorijski, registarski)
 - Kompleksne instrukcije (sin, cos, ...)
 - Mali broj registara opšte namene



Registri opšte namene

- Registri opšte namene:
 - 32-bitni: **eax**, **ebx**, **ecx**, **edx**
 - 16-bitni: **ax**, **bx**, **cx**, **dx**
 - 8-bitni: **ah**, **al**, **bh**, **bl**, **ch**, **cl**, **dh**, **dl**
- Registri opšte namene za nizove:
 - 32-bitni: **esi** (*source index*), **edi** (*destination index*)
 - 16-bitni: **si**, **di**
- Registri opšte namene za stek:
 - 32-bitni: **ebp** (*base pointer*), **esp** (*stack pointer*)
 - 16-bitni: **bp**, **sp**

Registri opšte namene

31	16	15	8	7	0	16-bit	32-bit
		ah		al		ax	eax
		bh		bl		bx	ebx
		ch		cl		cx	ecx
		dh		dl		dx	edx
		bp					ebp
		si					esi
		di					edi
		sp					esp

Segmentni registri (u sebi sadrže bazne adrese segmenata memorije)



Sadrže bazne adrese segmenata memorije:

- **CS** (code segment) sadrži adresu početka segmenta naredbi
- **DS** (data segment) sadrži adresu početka segmenta podataka
- **SS** (stack segment) sadrži adresu početka segmenta steka
- **es, fs i gs** (rukovanje podacima) su dodatni segmentni registri

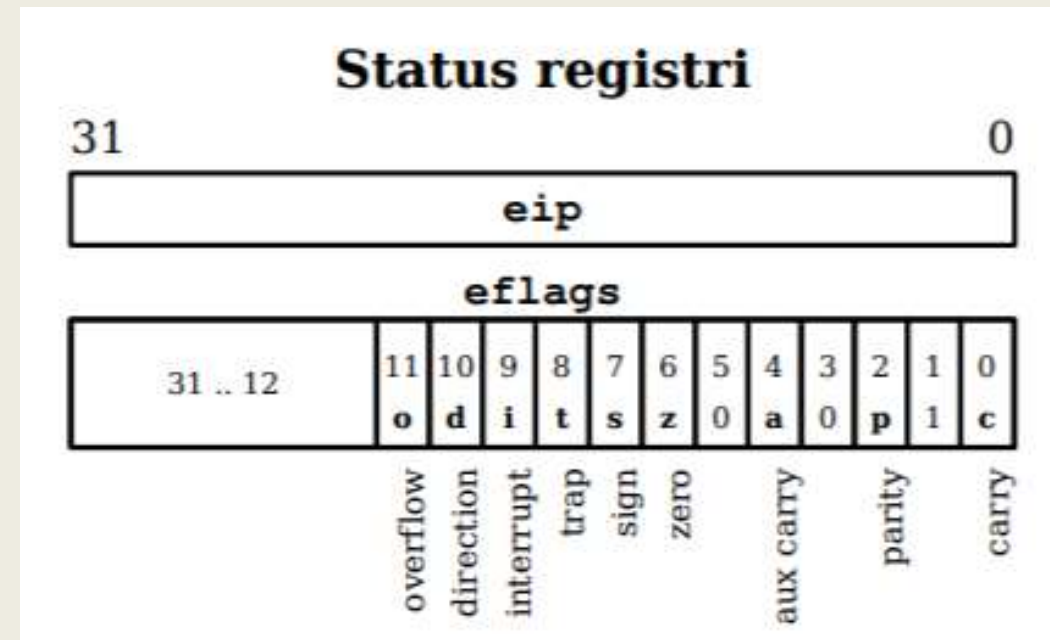
Segmentni registri

15	0
CS	
SS	
DS	
ES	
FS	
GS	

Statusni registri (indikatorski)

- Dostupni su samo pomoću specijalnih naredbi
- **eip** (instruction pointer) sadrži adresu naredne instrukcije
- **eflags** sadrži indikatore (tzv. Flegove, logičke promenjive) čije vrednosti zavise od rezultata izvršavanja naredbi:

- **c** (carry – prenos - nosilac)
- **o** (overflow - prekoračenje)
- **z** (zero - nula)
- **s** (sign - znak)
- **d** (direction - pravac)



Indikatori prekoračenja (c/o), nule i znaka (z/s)



Indikatori omogućavaju programu da koristi i označene i neoznačene brojeve

- **c (carry)** je indikator prekoračenja za **neoznačene brojeve**. Postavlja se na vrednost 1 ukoliko je prilikom izvršavanja operacije došlo do prenosa ili pozajmice (rezultat se posmatra kao neoznačen broj), a u suprotnom na 0 (nula).
- **o (overflow)** je indikator prekoračenja za **označene brojeve**. Postavlja se na vrednost 1 ukoliko je prilikom izvršavanja operacije došlo do prenosa ili pozajmice (rezultat se posmatra kao označen broj), a u suprotnom na 0 (nula).
- **z (zero)** je indikator nule. Postavlja se na vrednost 1 ukoliko je prilikom izvršavanja operacije rezultat jednak nuli, a u suprotnom na 0.
- **s (sign)** je indikator znaka. Postavlja se na vrednost 1 ukoliko je prilikom izvršavanja neke operacije najviši bit rezultata jednak jedinici (tj. ako rezultat (označeni broj) ima negativnu vrednost), a u suprotnom na 0.

Sintakse assemblera za familiju procesora x86



- Mašinski jezik Intelovog procesora je jedinstven, ali postoji više različitih asemblerskih dijalekata, koji se razlikuju po:
 - redosledu navođenja operanada,
 - sintaksi memorijskih, registarskih i neposrednih operanada,
 - po nazivima instrukcija.
- Neki od poznatijih assemblera su **GNU Assembler** (GAS) i **NASM**.
- Za familiju procesora x86 postoje dve sintakse pisanja naredbi:
 - **jedna je AT&T a**
 - **druga Intel-ova**

AT&T sintaksa



- ✓ Osnovni format:
mnemonik izvor, odredište
- ✓ Registri: prefiks „%“
- ✓ Neposredni operandi: prefiks „\$“
- ✓ Veličina operanda: mnemonik sufiks „b“, „w“ ili „l“ (malo L)
- ✓ Labela: ime labele + „:“

C program i Asemblerski program u AT&T sintaksi



```
#include <stdio.h>
int main(void) {
    int a,b,c;

    a=5;
    b=3;

    b=b+a;
}
```

.section .data	direktiva tj. pseudonaredba
.section .text	direktiva
.globl main	spoljašnja referenca
main:	labela main
movl \$5, %eax	mnemonik izvor, odredište
movl \$3, %ebx	mnemonik izvor, odredište
addl %eax, %ebx	registri prefiksi %
kraj:	
movl \$1, %eax	long, 32 bita
int \$0x80	

AT&T sintaksa



assembler	objašnjenje	C program
movl \$12,%eax	#12->eax	a = 12;
movl \$8,%ebx	#8->ebx	b = 8;
uporedi:	#	
cmpl %ebx,%eax	#	while (a != b)
je kraj	#skok ako eax=ebx	
cmpl %ebx,%eax	#	if (a > b)
ja vece	#skok ako eax>ebx	
subl %eax,%ebx	#ebx-eax->ebx	a = a - b;
jmp uporedi	#	
vece:	#	else
subl %ebx,%eax	#eax-ebx->eax	b = b - a;
jmp uporedi	#	
kraj:		
movl \$1, %eax		
movl \$0, %ebx		
int \$0x80		

mov - Smeštanje izvornog operanda u odredišni. U ovom programu se koriste registarski i neposredni operandi. Na primer, `movl 12, %eax` će smestiti broj 12 (neposredni operand, kodiran kao deo naredbe) u registar `eax`. Naredba `mov` ne utiče na indikatore.

cmp - Poređenje operanada. Tehnički, ovom naredbom se od odredišnog operanda oduzima izvorni, ali se pri tom sadržaj odredišta ne menja, već se samo ažuriraju indikatori: u ovom slučaju zanimaju nas indikator `z` (*zero*), koji se postavlja ako je rezultat nula, i indikator `c` (*carry*), koji se postavlja u slučaju prenosa sa najznačajnije pozicije.

je, ja, jmp - Relativni skokovi. Prva dva su uslovna. Do skoka **je** (*jump if equal*) dolazi ako važi `z=1`, a do skoka **ja** (*jump if above*) dolazi ako važi `c=0` i `z=0`. Naredba `jmp` je bezuslovni skok. Naredbe skoka ne utiču na indikatore.

sub - Oduzimanje izvornog operanda od odredišnog. Rezultat se smešta u odredište. Prilikom oduzimanja se ne uzima u obzir vrednost indikatora `c`. Naredba `sub` menja indikatore.

1. Smesti vrednost 12 u promenljivu `a`
2. Smesti vrednost 8 u promenljivu `b`
3. Uporedi `a` i `b`. Ako su jednaki, pređi na korak 9
4. Uporedi `a` i `b`. Ako je `a` veće od `b`, pređi na korak 7, inače pređi na sledeći korak
5. Oduzmi `a` od `b` i rezultat smesti u `b`
6. Pređi na korak 3
7. Oduzmi `b` od `a` i rezultat smesti u `a`
8. Pređi na korak 3
9. Kraj algoritma

Numerički tipovi podataka u C-u i assembleru u AT&T sintaksi



Velčina (biti)	Opseg	C	Assembler	Dibager
8	-128 do +127	char	.byte	bytes
8	0 do 255	unsigned char	.byte	bytes
16	-32768 do +32767	short int	.word	halfwords
16	0 do 65535	unsigned short int	.word	halfwords
32	-2147483648 do 2147483647	int	.long	words
32	0 do 4,294,967,295	unsigned int	.long	words
64	-2^{63} do $+2^{63}-1$	long long	.quad	giants
64	0 do $+2^{64}-1$	unsigned long long	.quad	giants

NASM assembler – Intelova sintaksa



- Sintaksa odgovara onoj koja se koristi u zvaničnoj Intelovoj dokumentaciji.
- **Odredišni** operand se uvek navodi **prvi**.
- Ispred registarskih i neposrednih operanada se ne navode nikakvi prefiksi
- **Memorijski** operandi se uvek navode u **uglastim** zagradama ([x], [y], [z+8], [d + 5 * k]...).
- Ukoliko instrukcija koristi bar jedan registarski operand, tada on implicitno određuje širinu operanada sa kojima se radi. U suprotnom, neophodno je ispred memorijskog operanda navesti **prefiks** kojim će se eksplicitno navesti širina operanada (**byte-8, word-16, dword-32, qword-64**).

Vrste instrukcija



Aseblerski jezik za Intel 80386 podržava više tipova instrukcija:

1. **aritmetičko-logičke**
2. **za pomeranje**
3. **za transfer**
4. **za poređenje**
5. **skoka**
6. **za rad sa potprogramima**

Aritmetičko-logičke instrukcije



Instrukcija	Opis	Operacija
ADD dest, src	<u>Sabira</u> src i dest i zbir smešta u dest	$\text{dest} = \text{dest} + \text{src}$
SUB dest, src	<u>Oduzima</u> src od dest i razliku smešta u dest	$\text{dest} = \text{dest} - \text{src}$
NEG op	<u>Promena znaka</u> operanda (potpuni komplement)	$\text{op} = -\text{op}$
(I)MUL src	<u>Množi</u> EAX registar sa src i proizvod smešta u EDX:EAX. IMUL – označeni, MUL – neoznačeni	$\text{EDX:EAX} = \text{EAX} * \text{src}$
(I)DIV src	<u>Deli</u> EDX:EAX registar sa src i količnik smešta u EAX, a ostatak u EDX. IDIV – označeni, DIV – neoznačeni	$\text{EAX} = \text{EDX:EAX} / \text{src}$ $\text{EDX} = \text{ostatak}$
AND dest, src	Bitovska konjunkcija dest i src, rezultat se smešta u dest	$\text{dest} = \text{src} \text{ and } \text{dest}$
OR dest, src	Bitovska disjunkcija dest i src, rezultat se smešta u dest	$\text{dest} = \text{src} \text{ or } \text{dest}$
XOR dest, src	Bitovska ekskluzivna disjunkcija dest i src, rezultat se smešta u dest	$\text{dest} = \text{src} \text{ xor } \text{dest}$
NOT op	Bitovska negacija operanda op (nepotpuni komplement)	$\text{op} = \text{not}(\text{op})$

Napomene: MUL



- MUL je instrukcija za **neoznačeno**, a IMUL za **označeno množenje**.
- **Operand ne može biti konstanta!**

$$n * n = 2n \text{ (} n - \text{ širina operanda)}$$

- Množenjem dva 32-bitna broja, dobija se 64-bitni rezultat!

$$\text{EDX:EAX (64-bit)} = \text{EAX(32-bit)} * \text{src(32-bit)}$$

- Viši deo proizvoda se smešta u EDX, a niži u EAX
- Instrukcija MUL podržava i **16-bitno** množenje i tada se koriste registri **AX** i **DX** umesto EAX i EDX.
- **Širina operanda *src*** određuje da li se radi o 16-bitnom ili 32-bitnom množenju. Ako se koristi memorijski operand, obavezno je navesti prefiks ***word*** za 16-bitno, tj. ***dword*** za 32-bitno množenje.

Napomene: DIV



- DIV je instrukcija za **neoznačeno**, a IDIV za **označeno deljenje**.
- **Operand ne može biti konstanta!**
- Pre operacije deljenja, neophodno je proširiti deljenik iz EAX registra na EDX:EAX:
 1. **NEOZNAČENO** deljenje: EDX proširiti **nulama** (0)
 2. **OZNAČENO** deljenje: EDX proširiti **znakom** (CDQ instrukcijom)
- Instrukcija DIV podržava i **16-bitno** deljenje i tada se koriste registri **AX** i **DX** umesto EAX i EDX.
- **Širina operanda src** određuje da li se radi o 16-bitnom ili 32-bitnom deljenju. Ako se koristi memorijski operand, obavezno je navesti prefiks **word** za 16-bitno, tj. **dword** za 32-bitno deljenje.

ZADATAK 1



- Napisati asemblerski program za sabiranje dva 32-bitna broja x i y.

$$Z = X + Y$$

```
1  %include "io.inc"
2
3  section .data
4  x: dd 0 ; prvi sabirak
5  y: dd 0 ; drugi sabirak
6  z: dd 0 ; rezultat (zbir)
7
8  section .text ; Sekcija sa instrukcijama
9  global CMAIN
10 CMAIN:
11
12     ; Ucitavamo podatke x i y sa standardnog ulaza
13     GET_DEC 4, [x]
14     GET_DEC 4, [y]
15
16     ; Ucitavamo x i y u registre eax i ecx procesora.
17     mov eax, [x]
18     mov ecx, [y]
19
20     add eax, ecx ; Sabiramo eax i ecx, i zbir smestamo u eax.
21     mov [z], eax ; Cuvamo rezultat u promenljivoj z u memoriji
22
23     ; Prikazujemo rezultat
24     PRINT_DEC 4, [z]
25
26     xor eax, eax
27     ret
```

SASM

File Edit Build Debug Settings Help

1.asm

```
1  %include "io.inc"
2  section .data
3
4  x: dd 0
5  y: dd 0
6  z: dd 0
7
8
9  section .text
10 global CMAIN
11 CMAIN:
12
13     GET_DEC 4, [x]
14     GET_DEC 4, [y]
15
16     mov eax, [x]
17     mov ecx, [y]
18
19     add eax, ecx
20     mov [z], eax
21
22     PRINT_DEC 4, [z]
23
24     xor eax, eax
25     ret
```

Input

5
10

Output

15

[21:28:59] Built successfully.

[21:28:59] The program is executing...

[21:29:00] The program finished normally. Execution time: 0.076 s

[21:29:51] Build started...

[21:29:52] Built successfully.

[21:29:52] The program is executing...

[21:29:52] The program finished normally. Execution time: 0.088 s

Instrukcije za transfer



- Za početak ćemo koristiti samo instrukciju **MOV** za kopiranje sadržaja iz **src** u **dest**.
- Ostale instrukcije ćemo objasniti kada budemo pisali složenije programe

Instrukcija	Opis	Napomena
MOV dest, src	<u>Kopira</u> src u dest	-
PUSH op	Postavlja podatak op na stek	op može biti veličine 1, 2, 4 ili 8 bajta. Automatski se menja pokazivač steka (ESP registar)
POP op	Skida podatak sa steka i smešta ga u op	op ne može biti konstanta. Automatski se menja pokazivač steka (ESP registar)
LEA dest, mem_op	Prebacuje adresu memorijskog operanda mem_op u dest	Kada treba da pristupimo adresi memorijskog operanda (npr. da bismo je preneli funkciji kao argument)
XCHG op1, op2	Razmenjuje vrednosti operanada	Ne može koristiti neposredne operande (konstante)

ZADATAK 1b

- Napisati program koji sa ulaza preuzima 3 vrednosti i smešta ih u promenljive a , b i c , računa vrednost traženog izraza i rezultat smešta u promenljivu d , čiju vrednost na kraju ispisuje na izlazu.



The screenshot shows the SASM (Simple Assembler) interface. The main window displays the assembly code for a program named 1b.asm. The code includes a data section with three 4-byte variables (x, y, z, d) initialized to 0. The text section contains the main logic: reading three integers from the input, storing them in x, y, and z, then calculating their sum and storing the result in d. Finally, it prints the value in d and exits.

```
1  %include "io.inc"
2  section .data
3
4  x: dd 0
5  y: dd 0
6  z: dd 0
7  d: dd 0
8
9
10 section .text
11 global CMAIN
12 CMAIN:
13
14     GET_DEC 4, [x]
15     GET_DEC 4, [y]
16     GET_DEC 4, [z]
17
18     mov eax, [x]
19     mov ebx, [y]
20     mov ecx, [z]
21
22     add eax, ebx
23     add ecx, eax
24     mov [d], ecx
25
26     PRINT_DEC 4, [d]
27
28     xor eax, eax
29     ret
```

On the right side of the interface, there are two panels. The 'Input' panel shows the input values: 1, 14, and 1. The 'Output' panel shows the result of the calculation: 16.

Zadatak 2:



- a) Učitati vrednosti 5 i 8 u registre ***eax*** i ***ebx***, a zatim uraditi kopiranje u registre ***ecx*** i ***edx***:

$$ecx = eax$$
$$edx = ebx$$
- b) Pokrenite program u **Debug** režimu i pratite stanje registara nakon svake izvršene instrukcije.
- c) Dodajte učitavanje vrednosti sa standardnog ulaza i ispis vrednosti sva 4 registra na standardni izlaz.

SASM

File Edit Build Debug Settings Help

Memory

Variable or expression	Value	Type
Add variable...	Smart	d Array size

*12-01-mov.asm

```
1 %include "io.inc"
2
3 section .text
4 global CMAIN
5 CMAIN:
6     mov ebp, esp; for correct debugging
7     mov eax, 5
8     mov ebx, 8
9     mov ecx, eax
10    mov edx, ebx
11
12 → xor eax, eax
13    ret
```

SASM

File Edit Build Debug Settings Help

Memory

Variable or expression	Value	Type
Add variable...	Smart	d Array size

Continue F5

Step over F10

Step into F11

Toggle breakpoint F8

Show registers Ctrl+R

Show memory Ctrl+M

Stop

Input

Output

Registers

Register	Hex	Info
eax	0x5	5
ecx	0x5	5
edx	0x8	8
ebx	0x8	8
esp	0x61110c	0x61110c
ebp	0x61ff0c	0x61ff0c
esi	0x40126c	4199020
edi	0x40126c	4199020
eip	0x4013a0	0x4013a0 <main+16>
eflags	0x202	[IF]
cs	0x23	35
ss	0x2b	43
ds	0x2b	43
es	0x2b	43
fs	0x53	83
gs	0x2b	43

[22:57:47] Built successfully.

[22:57:47] The program is executing...

[22:57:47] The program finished normally. Execution time: 0.085 s

[22:57:54] Build started...

[22:57:54] Built successfully.

[22:57:54] Debugging started...

GDB command:

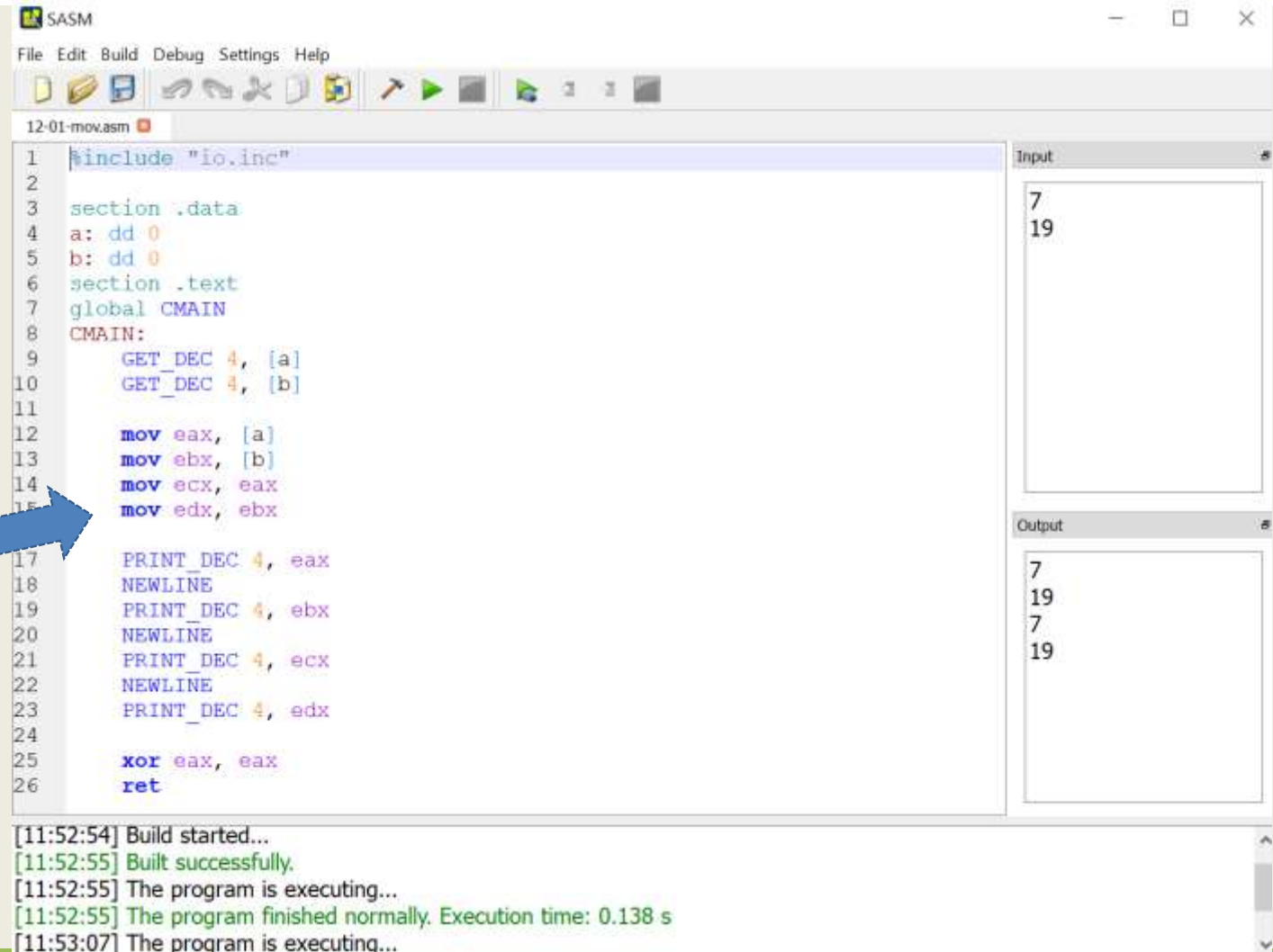
Print

Perform

Zadatak 3:

- **STDIN/OUT**

- Alociranje podatka određene veličine sa inicijalnom vrednošću 0:
 - **db** – 1 bajt
 - **dw** – 2 bajta
 - **dd** – 4 bajta
 - **dq** – 8 bajta
- **Labele** a: i b: predstavljaju **adrese** na kojima će biti smešteni alocirani podaci u memoriji.
- Ukoliko želimo da koristimo podatke iz memorije, **memorijske operande** navodimo u uglastim zagradama: [a], [b]. **Bez zagrada bi učitali njihovu adresu!**



```
1  %include "io.inc"
2
3  section .data
4  a: dd 0
5  b: dd 0
6  section .text
7  global CMAIN
8  CMAIN:
9      GET_DEC 4, [a]
10     GET_DEC 4, [b]
11
12     mov eax, [a]
13     mov ebx, [b]
14     mov ecx, eax
15     mov edx, ebx
16
17     PRINT_DEC 4, eax
18     NEWLINE
19     PRINT_DEC 4, ebx
20     NEWLINE
21     PRINT_DEC 4, ecx
22     NEWLINE
23     PRINT_DEC 4, edx
24
25     xor eax, eax
26     ret
```

[11:52:54] Build started...
[11:52:55] Built successfully.
[11:52:55] The program is executing...
[11:52:55] The program finished normally. Execution time: 0.138 s
[11:53:07] The program is executing...

Zadatak 4:



- Napisati program u asemblerskom jeziku koji računa vrednost izraza:

$$r = 2 * a * b + c / d - e / 2$$

- Na kraju programa proveriti da li je **rezultat paran ili neparan**, a zatim na standardni izlaz ispod rezultata ispisati:
 - **1 ako je rezultat neparan broj**
 - **0 ako je rezultat paran broj**
- Pokrenuti program u **Debug** režimu i pratiti ponašanje registara i memorije.



*4.asm

```
1  %include "io.inc"
2  section .data
3  a: dd 0
4  b: dd 0
5  c: dd 0
6  d: dd 0
7  e: dd 0
8  r: dd 0
9  section .text
10 global CMAIN
11 CMAIN:
12     GET_DEC 4, [a]
13     GET_DEC 4, [b]
14     GET_DEC 4, [c]
15     GET_DEC 4, [d]
16     GET_DEC 4, [e]
17     mov eax, [a]
18     mov ebx, [b]
19     mov ecx, 2
20     mul ebx
21     mul ecx
22     mov ebx, eax
23     mov eax, [c]
24     mov ecx, [d]
25     mov edx, 0
26     div ecx
27     add ebx, eax
28     mov eax, [e]
29     mov ecx, 2
30     mov edx, 0
31     div ecx
32     sub ebx, eax
33     PRINT_DEC 4, ebx
34     NEWLINE
35     mov eax, ebx
36     mov ebx, 2
37     mov edx, 0
38     div ebx
39     PRINT_DEC 4, edx
40     NEWLINE
41     xor eax, eax
42     ret
```

Input

1
0
20
4
1

Output

5
1

Zadatak 5



- Realizovati izraz iz prethodnog zadatka:

$$r = 2*a*b + c/d - e/2$$

- Takođe proveriti da li je broj **paran** ili **neparan** aritmetičkim ili, bolje, logičkim operacijama i rezultat ispisati na standardni izlaz ispod rezultata:
 - 1 ako je rezultat neparan broj
 - 0 ako je rezultat paran broj
- Napraviti testni slučaj (zadati ulazne operande a, b, c, d i e) tako da rezultat bude vaš redni broj u indeksu.
- Napraviti **printscreen** ekrana nakon pokrenutog programa.
- *Rešenje zadatka (.asm i printscreen) poslati na Teams (napravite svoj folder u folderu Vežbe studenata).*

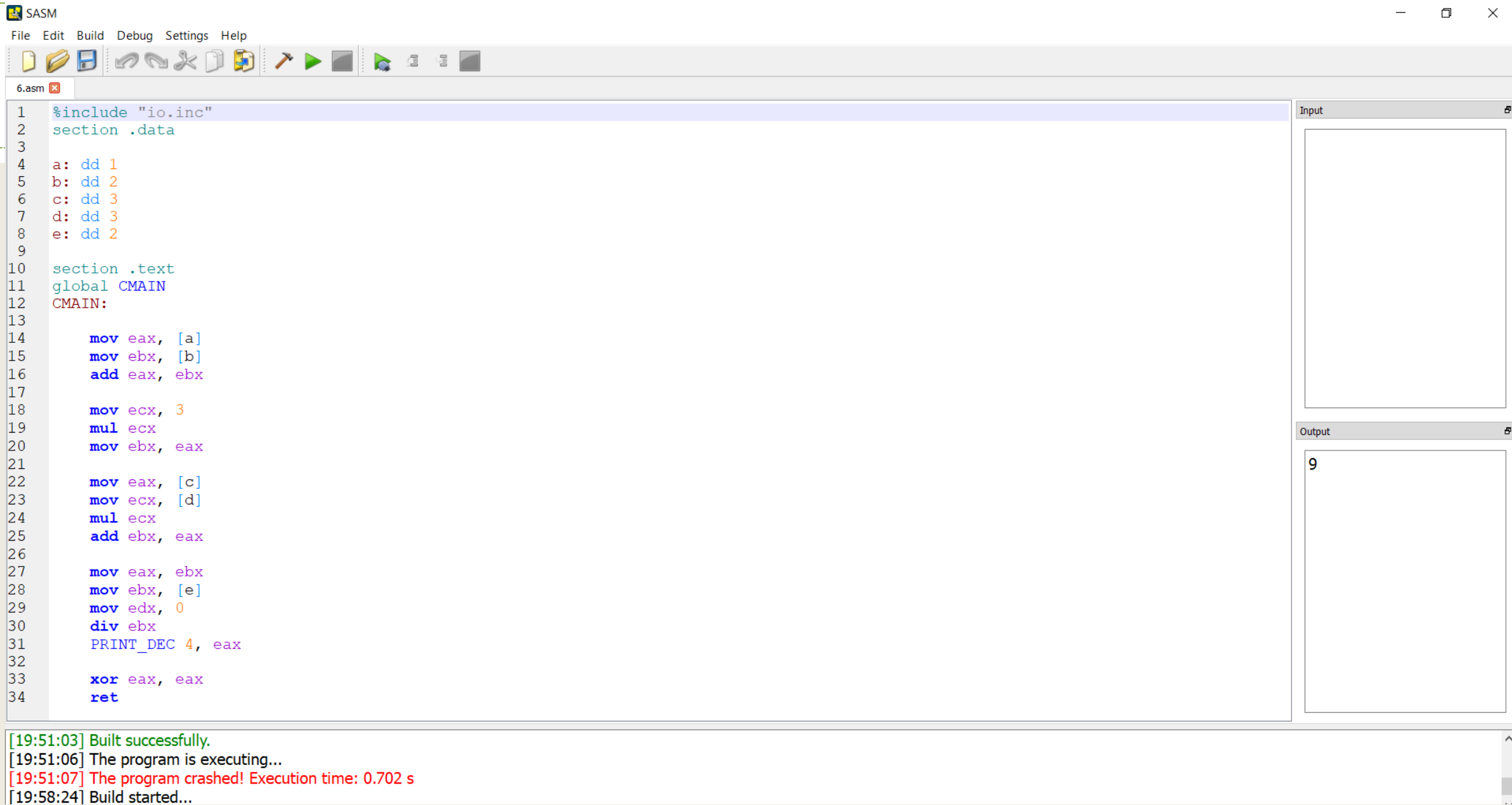
Zadatak 6



- Realizovati izraz:

$$r = (3 * (a + b) + c * d) / e$$

- **Napraviti testni slučaj (zadati ulazne operande a, b, c, d i e) tako da rezultat bude vaš redni broj u indeksu.**
- **Napraviti `printscreen` ekrana nakon pokrenutog programa.**
- ***Rešenje zadatka (.asm i `printscreen`) poslati na Teams (napravite svoj folder u folderu Vežbe studenata).***



Instrukcije pomeranja



Instrukcija	Opis	Operacija
SHL dest, src	<u>Pomera u levo</u> bitovski sadržaj dest operanda za broj bitova dat src operandom	$\text{dest} = \text{dest} \ll \text{src}$
SHR dest, src	<u>Pomera logički u desno</u> bitovski sadržaj dest operanda za broj bitova dat src operandom	$\text{dest} = \text{dest} \gg \text{src}$
SAR dest, src	Pomera aritmetički u desno bitovski sadržaj dest operanda za broj bitova dat src operandom	$\text{dest} = \text{dest} \gg \text{src}$

- Drugi operand instrukcije pomeranja može biti:
 - **8-bitna konstanta**
 - **registar CL**

Množenje i deljenje – pomeranjem



- Instrukcije pomeranja se mogu koristiti za efikasno množenje i deljenje stepenom dvojke:
- Množenje sa 2^n = pomeranje ulevo za n mesta
- Neoznačeno deljenje sa 2^n = logičko pomeranje udesno za n mesta
- Označeno deljenje sa 2^n = aritmetičko pomeranje udesno za n mesta
- Kad god je moguće, koristite instrukcije za pomeranje umesto MUL i DIV instrukcija!

Zadatak 7:



- Realizovati izraz iz prethodnog zadatka korišćenjem instrukcija za pomeranje (tamo gde je moguće):

$$r = 2 * a * b + c / d - e / 2$$

- Takođe proveriti da li je broj **paran** ili **neparan korišćenjem instrukcija za pomeranje** i rezultat ispisati na standardni izlaz ispod rezultata:
 - **1 ako je rezultat neparan broj**
 - **0 ako je rezultat paran broj**

```
13  r: dd 0
14  section .text
15  global CMAIN
16  CMAIN:
17      GET_UDEC 4, [a]
18      GET_UDEC 4, [b]
19      GET_UDEC 4, [c]
20      GET_UDEC 4, [d]
21      GET_UDEC 4, [e]
22
23      mov eax, [a]
24      shl eax, 1      ; eax = a * 2
25      mul dword[b]    ; edx, eax = eax * b ----> (edx, eax = 2 * a * b)
26      mov ecx, eax    ; ecx = 2*a*b
27
28      mov eax, [c]
29      mov edx, 0
30      div dword[d]    ; eax = c/d
31      add ecx, eax    ; ecx = 2*a*b + c/d
32
33      mov eax, [e]
34      shr eax, 1      ; eax = e/2
35      sub ecx, eax    ; ecx = 2*a*b + c/d - e/2
36      mov [r], ecx
37
38      shl ecx, 31     ; paran ili neparan ? - provera pomeranjem
39      shr ecx, 31
40
41      PRINT_UDEC 4, [r]
42      NEWLINE
43      PRINT_UDEC 4, ecx
44
45      xor eax, eax
46      ret
47
```

Input

1
2
8
2
4

Output

6
0

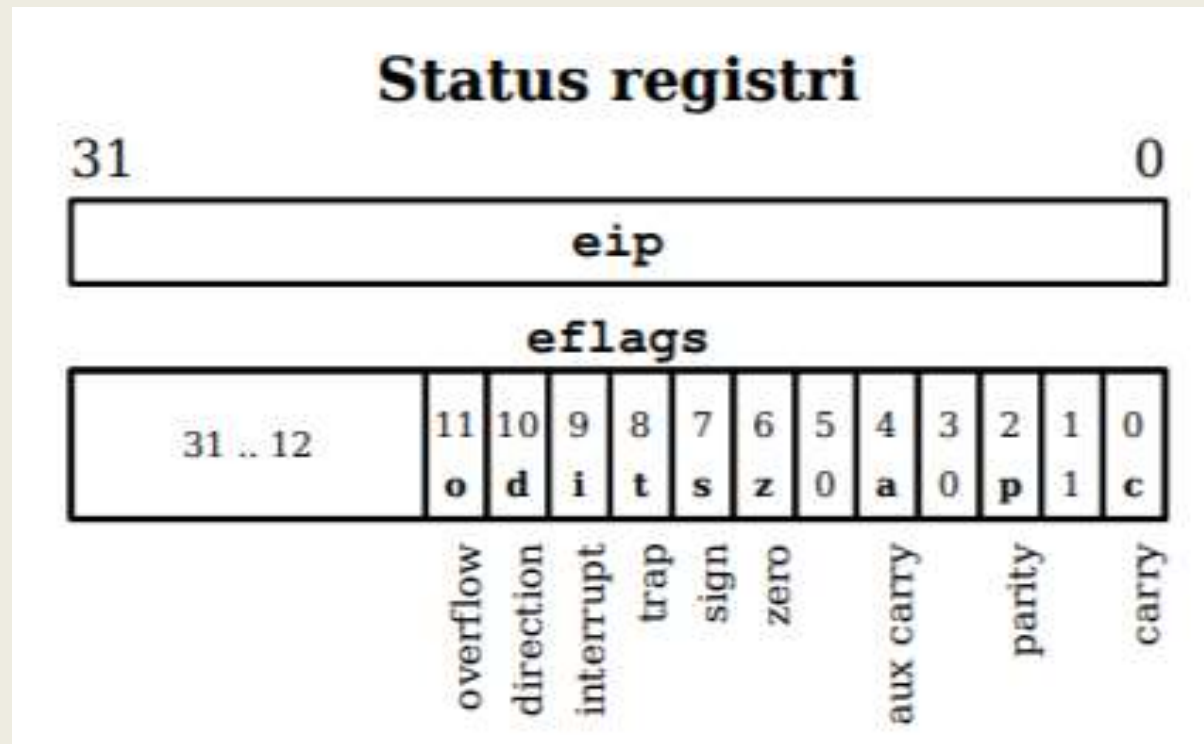
Instrukcije poređenja



Instrukcija	Opis	Napomena
CMP op1, op2	<u>Poređenje operanada</u> na osnovu RAZLIKE . Oduzima op2 od op1 , ali razliku nigde ne upisuje, već samo na osnovu rezultata ažurira statusne bite (flegove) u EFLAGS registru	<ul style="list-style-type: none">• Ako je $x - y < 0$, tada je $x < y$• Ako je $x - y = 0$, tada je $x = y$• Ako je $x - y > 0$, tada je $x > y$• Od rezultata nam treba samo ZNAK!
TEST op1, op2	Omogućava testiranje tj. poređenje pojedinačnih bita operanada op1 i op2. Računa bitovsku konjunkciju (AND) operanada op1 i op2, ali rezultat nigde ne upisuje, već samo na osnovu rezultata ažurira statusne bite (flegove) u EFLAGS registru	Primer korišćenja: <ul style="list-style-type: none">• Napravi se „maska“, tj. Vrednost koja sadrži jedinicu na bitskoj poziciji koja nas zanima, a NULE na ostalim pozicijama.• Primenimo instrukciju TEST nad datim podatkom i kreiranom maskom.• Rezultat je NULA, akko je odgovarajući bit u datom podatku jednak 0.

Instrukcije poređenja - eflags

- Instrukcije poređenja ne upisuju rezultat u registre opšte namene
- Menjaju stanje statusnih registara eip i eflags



Instrukcije skoka



Instrukcija	Opis	Jump if	Stanje flegova
JMP addr	Bezuslovni skok na adresu addr	-	UVEK SKAČE
JE addr	Skok na adresu addr , ako je jednako	equal	ZERO = 1
JA addr	Skok na adresu addr ako je veće (neoznačeno)	above	CARRY = 0 or ZERO = 0
JB addr	Skok na adresu addr ako je manje (neoznačeno)	below	CARRY = 1
JAE addr	Skok na adresu addr ako je veće ili jednako (neoznačeno)	above or equal	CARRY = 0
JBE addr	Skok na adresu addr ako je manje ili jednako (neoznačeno)	below or equal	CARRY = 1 or ZERO = 1
JG addr	Skok na adresu addr ako je veće (označeno)	greater	(SIGN and OVERFLOW) or ZERO = 0
JL addr	Skok na adresu addr ako je manje (označeno)	less	SIGN and OVERFLOW = 1
JGE addr	Skok na adresu addr ako je veće ili jednako (označeno)	greater or equal	SIGN and OVERFLOW = 0
JLE addr	Skok na adresu addr ako je manje ili jednako (označeno)	less or equal	(SIGN and OVERFLOW) or ZERO = 1

Instrukcije skoka



- Instrukcije skoka najčešće imaju **neposredni operand** koji predstavlja adresu na koju treba skočiti:
 - Najčešće se navodi kao **labela** u assembleru
 - Može biti i **registarski operand** i tada je adresa sadržana u tom registru
- Instrukcije uslovnog skoka obično slede nakon **CMP** ili **TEST** instrukcija, kojima se postavljaju statusni biti (flegovi)
- **NAPOMENA:** Potrebno je voditi računa da li su operandi **označeni** ili **neoznačeni** brojevi, jer se koriste različite instrukcije zbog različite kombinacije flegova koje daju ispunjen uslov!

Zadatak 8



- Napisati program u asemblerskom jeziku koji računa maksimum dva cela označena broja x i y. Rezultat **max** je potrebno upisati u memoriju i na standardni izlaz.

```
if (x > y)
```

```
    max = x;
```

```
else
```

```
    max = y;
```

SASM

File Edit Build Debug Settings Help

1b.asm 8.asm

```
1  $include "io.inc"
2  section .data
3  a: dd 0
4  b: dd 0
5  maximum: dd 0
6
7  section .text
8  global CMAIN
9  CMAIN:
10
11     GET_DEC 4, [a]
12     GET_DEC 4, [b]
13
14     mov eax, [a]
15     mov ebx, [b]
16
17     cmp eax, ebx
18     jg veceodnule
19     mov [maximum], ebx
20     jmp rezultat
21
22     veceodnule:
23     mov [maximum], eax
24
25     rezultat:
26     PRINT_DEC 4, [maximum]
27     NEWLINE
28
29     xor eax, eax
30     ret
```

Input

5
11

Output

11

[21:11:23] Built successfully.
[21:11:25] The program is executing...
[21:11:25] The program finished normally. Execution time: 0.057 s
[21:13:58] Build started...
[21:13:58] Built successfully.
[21:14:00] The program is executing...
[21:14:00] The program finished normally. Execution time: 0.059 s

Zadatak 9:



- Napisati program u asemblerskom jeziku koji određuje da li je zadati broj pozitivan ili negativan. Na standardnom izlazu ispisati string u zavisnosti od rezultata: POZITIVAN ili NEGATIVAN.
- Takođe, proveriti da li je broj paran ili neparan korišćenjem TEST instrukcije. Na standardnom izlazu ispisati string PARAN ili NEPARAN, u zavisnosti od vrednosti zadatog broja.



1b.asm x 8.asm x 9.asm x

```
1  %include "io.inc"
2
3  section .data
4  a: dd 0
5
6  section .text
7  global CMAIN
8  CMAIN:
9
10     GET_DEC 4, [a]
11     mov eax, [a]
12
13     cmp eax, 0
14
15     jl negativan
16
17     PRINT_STRING "pozitivan"
18     NEWLINE
19     jmp dalje
20
21     negativan:
22     PRINT_STRING "negativan"
23     NEWLINE
24
25     dalje:
26     test eax, 1
27     je paran
28
29     PRINT_STRING "neparan"
30     NEWLINE
31     jmp kraj
32
33     paran:
34     PRINT_STRING "paran"
35     NEWLINE
36
37     kraj:
38
39     xor eax, eax
40     ret
```

Input

-10

Output

negativan
paran

IF-ELSE primer u assembleru



- C pseudo kod

```
if (eax == 0)
    ebx = 1;
else
    ebx = 2;
```

- Kod u asemblerskoj jeziku

```
cmp eax, 0      ; ZF=1 ako je eax-0=0
je then         ; ako je ZF=1, skoči na then
mov ebx, 2      ; ELSE deo IF-a
jmp kraj        ; preskoči THEN deo
then:
    mov ebx, 1   ; THEN deo IF-a
kraj:
```

IF-ELSE konstrukcija



- C pseudo kod

```
if (uslov)
    then blok;
else
    else blok;
```

- Kod u asemblerskoj jeziku

```
; poređenje registara (flags)
jxx else
; kod then bloka
jmp kraj
else:
; kod else bloka
kraj:
```

- jxx se postavlja tako da je if uslov false!

Zadatak 10

- Napisati program u asemblerskom jeziku koji računa vrednost y prema sledećem izrazu:

$$y = \begin{cases} x - 1, & x < 3 \\ x + 3, & \text{za ostale slučajeve} \end{cases}$$

```
1  %include "io.inc"
2
3  ; Program računa vrednost y prema izrazu:
4  ; y = x-1, x<3
5  ; y = x+3, inače
6
7  section .data
8  y: dd 0
9  section .text
10 global CMAIN
11 CMAIN:
12     GET_DEC 4, eax
13     cmp eax, 3
14     jge plus_tri
15     sub eax, 1      ; eax = x - 1
16     jmp kraj
17
18 plus_tri:
19     add eax, 3      ; eax = x + 3
20 kraj:
21     mov [y], eax
22     PRINT_DEC 4, eax
23
24     xor eax, eax
25     ret
```

Input

-2

Output

-3

Zadatak 11



- Napisati program u asemblerskom jeziku koji učitava prirodan broj. Ako je broj paran, onda treba izračunati i ispisati njegovu polovinu, a ako je neparan ispisati kvadrat njegovog prethodnika.

$$y = \begin{cases} \frac{a}{2}, & \text{ako je } a \text{ paran broj} \\ (a - 1)^2, & \text{ako je } a \text{ neparan broj} \end{cases}$$



```
1  %include "io.inc"
2  ; Napisati program u asemblerskom jeziku koji učitava prirodan broj.
3  ; Ako je broj paran, onda izračunati i ispisati njegovu polovinu,
4  ; a ako je neparan ispisati kvadrat njegovog prethodnika.
5
6  section .data
7  a: dd 0
8  y: dd 0
9  section .text
10 global CMAIN
11 CMAIN:
12     GET_UEC 4, [a]
13     mov eax, [a]
14     and eax, 1
15     je polovina      ; ako je paran skoči -> izračunaj polovinu
16 kvadrat:
17     PRINT_STRING "NEPARAN BROJ -> (a-1)*(a-1) = "
18     mov eax, [a]
19     sub eax, 1        ; eax = a - 1
20     mul eax           ; edx:eax = eax * eax = (a - 1) * (a - 1)
21
22     jmp ispis
23 polovina:
24     PRINT_STRING "PARAN BROJ -> a/2 = "
25     mov eax, [a]
26     mov edx, 0
27     mov ecx, 2
28     div ecx
29 ispis:
30     PRINT_DEC 4, eax
31     xor eax, eax
32     ret
```

Input

9

Output

NEPARAN BROJ -> (a-1)*(a-1) = 64

WHILE petlja



- C pseudo kod

```
while (uslov) {  
    telo petlje;  
}
```

- Kod u asemblerskoj jeziku

```
while:  
    ; postavljanje statusnih bita  
    jxx endwhile  
    ; telo petlje  
    jmp while  
endwhile:
```

- jxx se postavlja tako da je uslov == false!

Zadatak 12



- Napisati program u asemblerskom jeziku koji računa zbir svih brojeva do zadatog broja. U nastavku je dat pseudo kod u programskom jeziku C.

```
while (i <= N) {  
    sum += i;  
    i++;  
}
```



```
1  %include "io.inc"
2
3  ; Program računa zbir svih brojeva do zadanog broja pomoću while petlje
4
5  section .text
6  global CMAIN
7  CMAIN:
8      GET_UDEC 4, eax
9      mov ebx, 0      ; i = 0
10     mov ecx, 0      ; sum = 0
11 while:
12     cmp ebx, eax     ; da li je i <= N
13     ja  kraj
14     add ecx, ebx     ; sum += i
15     add ebx, 1      ; i++
16     jmp while
17 kraj:
18     PRINT_STRING "Suma svih brojeva = "
19     PRINT_UDEC 4, ecx
20     xor eax, eax
21     ret
```

Input

10

Output

Suma svih brojeva = 55

DO-WHILE petlja



- C pseudo kod

```
do{  
    telo petlje;  
} while (uslov);
```

- Kod u asemblerskoj jeziku

```
do:  
    ; telo petlje  
    ; postavljanje statusnih bita  
jxx do
```

- jxx se postavlja tako da je uslov == true!

Zadatak 13



- Napisati program u asemblerskom jeziku koji ispisuje prvih N brojeva, $N > 0$. Vrednost N zadati preko standardnog ulaza. Koristiti DO-WHILE petlju. U nastavku je dat pseudo kod u programskom jeziku C.

```
if(N > 0) {  
    do {  
        print(„%d“, i);  
        i++;  
    } while (i != N)  
}
```

Zadatak 14



- Napisati program u asemblerskom jeziku koji računa zbir svih **PARNIH** i **NEPARNIH** brojeva manjih od N. Preko standardnog ulaza zadati vrednost N. U nastavku je dat pseudo kod u programskom jeziku C.

```
while (i <= N) {  
    if(i%2 == 0)  
        parni += i;  
    else  
        neparni += i;  
    i++;  
}
```

- Napraviti **printscreen** ekrana nakon pokrenutog programa.
- ***Rešenje zadatka poslati na Teams.***



SASM

File Edit Build Debug Settings Help

14 - domaći 2.asm

```
1  %include "io.inc"
2  section .data
3  a: dd 0
4  section .text
5  global CMAIN
6  CMAIN:
7
8      GET_DEC 4, [a]
9      mov eax, [a]
10     mov ebx, 0      ; brojač
11     mov ecx, 0      ; parni
12     mov edx, 0      ; neparni
13
14     petlja:
15     cmp ebx, eax
16     je kraj
17
18     inc ebx
19     test ebx, 1
20     je paran
21
22     add edx, ebx
23     jmp petlja
24
25     paran:
26     add ecx, ebx
27     jmp petlja
28
29     kraj:
30     PRINT_STRING "zbir parnih: "
31     PRINT_DEC 4, ecx
32     NEWLINE
33     PRINT_STRING "zbir neparnih: "
34     PRINT_DEC 4, edx
35
36     xor eax, eax
37     ret
```

Input

9

Output

zbir parnih: 20
zbir neparnih: 25