무료 전자 책

배우기 Assembly Language

Free unaffiliated eBook created from
Stack Overflow contributors.

#assembly

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: assembly-language

It is an unofficial and free Assembly Language ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official Assembly Language.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

1:

. CPU ( ) CPU CPU .

Assembly , HW . .

CPU CPU HW ASM .

:

X86

Examples

, . , 8 16 .

:

```
mov eax, 4
cmp eax, 5
je point
```

. C ++ . , ARM, MIPS, x86 . . x86 **gas** (`gas` GNU Assembler) **Intel** (x86) . .

. .

.

- CISC ( ) : .
- RISC (Reduced Instruction Set Computers) : , .

CISC RISC .

. *gcc* . .

" " , . . GNU *binutils* , . *gcc* .

C C ++ . . . , . ( ).) / .

, . GNU, sdcc, llvm .

CPU (Central Processing Unit) .

( ) (CPU), - (ROM / RAM) (I / O) .

( ) (:) .). . .

?

. CPU . CPU () CPU .

2 8 (8 1 . 0-255) : `60 201` , Zilog Z80 CPU : `INC a` (a) `RET` (CPU).

: , / `3C C9` (60 201 16). .

CPU .

```
subroutineIncrementA:
    INC   a
    RET

dataValueDefinedInAssemblerSource:
    DB    60          ; define byte with value 60 right after the ret
```

16 `3C C9 3C` , `3C C9 3C` : , ( ), ( )

60 "data" . CPU `INC a` . CPU .

(- ,) CPU .

CPU . . OS " " I / O (:).

( ) (CPU) / / .

(GCC : `gcc -S helloworld.c` C `gcc -S helloworld.c helloworld.c`).

Linux x86_64 (Intel 64) Hello world

```
section .data
    msg db "Hello world!",10      ; 10 is the ASCII code for a new line (LF)

section .text
    global _start

_start:
    mov rax, 1
    mov rdi, 1
    mov rsi, msg
    mov rdx, 13
    syscall

    mov rax, 60
    mov rdi, 0
    syscall
```

Netwide Assembler , `nasm` . . (`helloworld.asm`). , .

- `nasm -felf64 helloworld.asm`
- `ld helloworld.o -o helloworld`
- `./helloworld`

Linux `sys_write` . x86_64 syscalls . C .

```
#include <unistd.h>

#define STDOUT 1

int main()
{
    write(STDOUT, "Hello world!\n", 13);
    _exit(0);
}
```

( ) .

- `gcc helloworld_c.c -o helloworld_c .`
- `./helloworld_c`

OS X Hello World (x86_64,)

```
.intel_syntax noprefix

.data

.align 16
hello_msg:
    .asciz "Hello, World!"

.text

.global _main
_main:
    push rbp
    mov rbp, rsp

    lea rdi, [rip+hello_msg]
    call _puts

    xor rax, rax
    leave
    ret
```

:

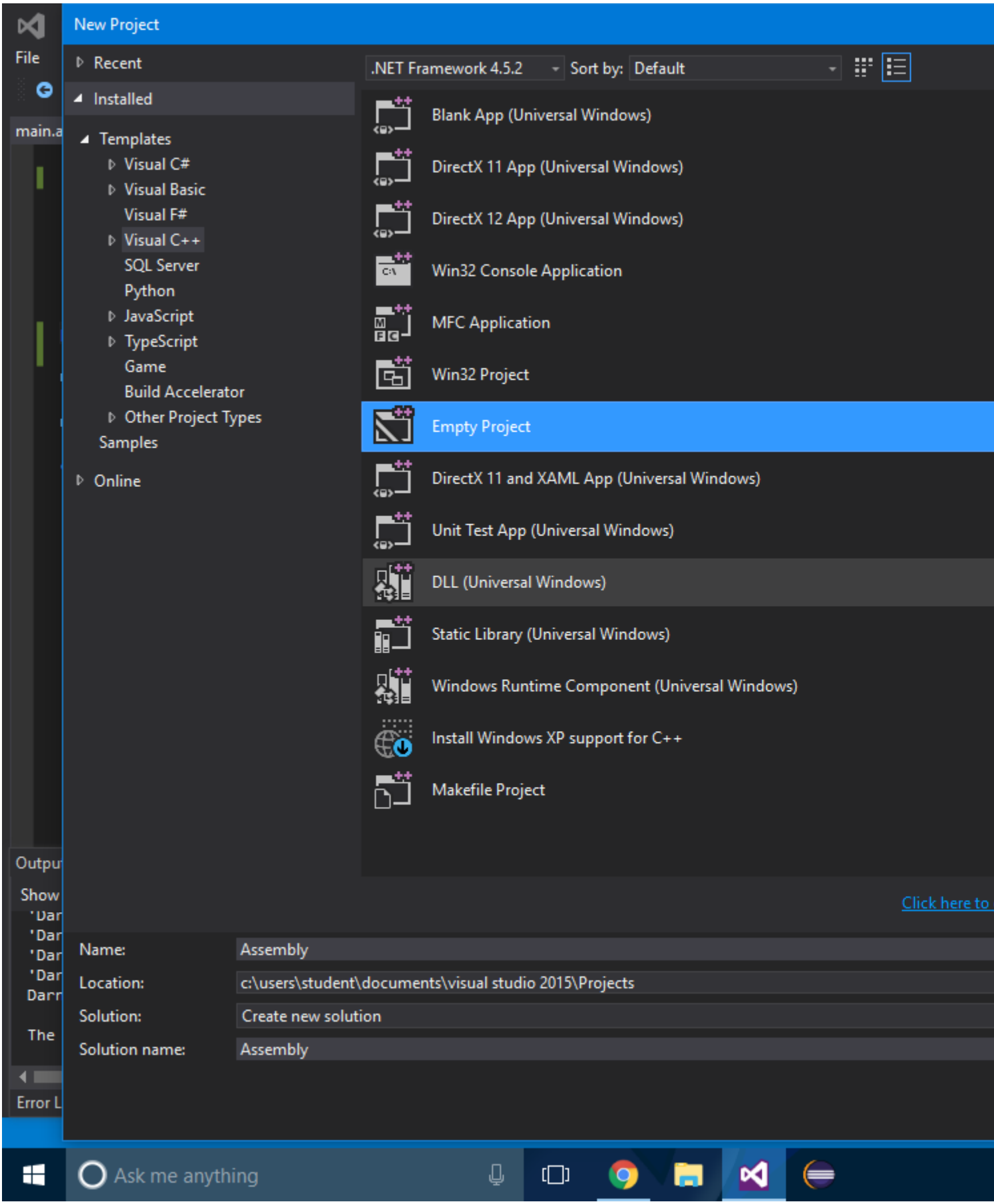
```
clang main.s -o hello
./hello
```

:

- OS X API . C . ( )
- 16 . (Intel)
- : rdi, rsi, rdx, rcx, r8 r9. (System V ABI)

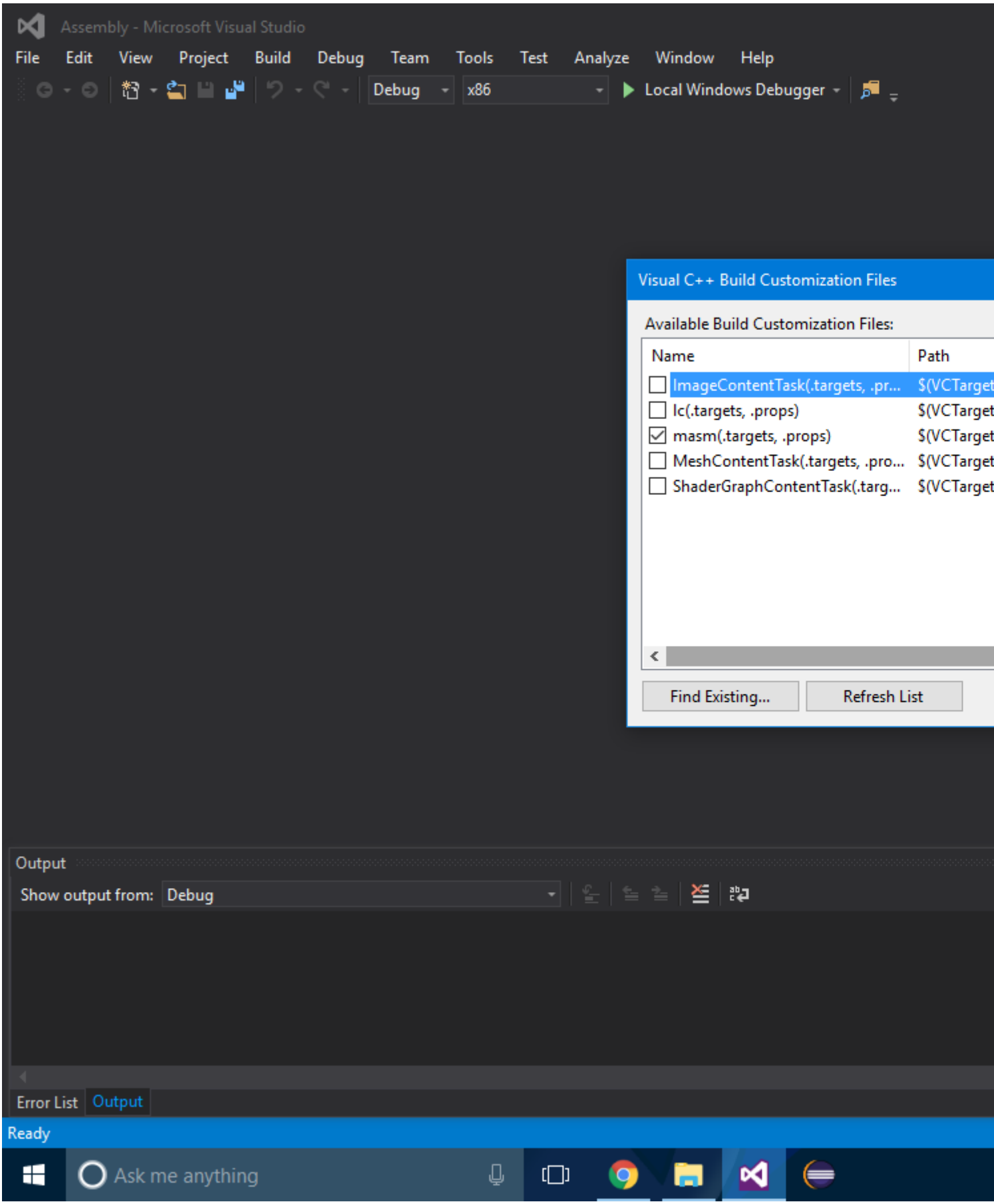
Visual Studio 2015 x86

1 : **->** .


New Project
File
Recent
Installed
Templates
Visual C#
Visual Basic
Visual F#
Visual C++
SQL Server
Python
JavaScript
TypeScript
Game
Build Accelerator
Other Project Types
Samples
Online
.NET Framework 4.5.2
Sort by:
Default
Blank App (Universal Windows)
DirectX 11 App (Universal Windows)
DirectX 12 App (Universal Windows)
Win32 Console Application
MFC Application
Win32 Project
Empty Project
DirectX 11 and XAML App (Universal Windows)
Unit Test App (Universal Windows)
DLL (Universal Windows)
Static Library (Universal Windows)
Windows Runtime Component (Universal Windows)
Install Windows XP support for C++
Makefile Project
Name:
Assembly
Location:
c:\users\student\documents\visual studio 2015\Projects
Solution:
Create new solution
Solution name:
Assembly
Ask me anything

2 : **->** .

3 : *".masm"* .


Assembly - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test Analyze Window Help
Debug
x86
Local Windows Debugger
Visual C++ Build Customization Files
Available Build Customization Files:
Name
Path
ImageContentTask(.targets, .pr...
$(VCTarget
lc(.targets, .props)
masm(.targets, .props)
MeshContentTask(.targets, .pro...
ShaderGraphContentTask(.targ...
Find Existing...
Refresh List
Output
Show output from: Debug
Error List
Output
Ready
Ask me anything

4 : " " .

5 : .

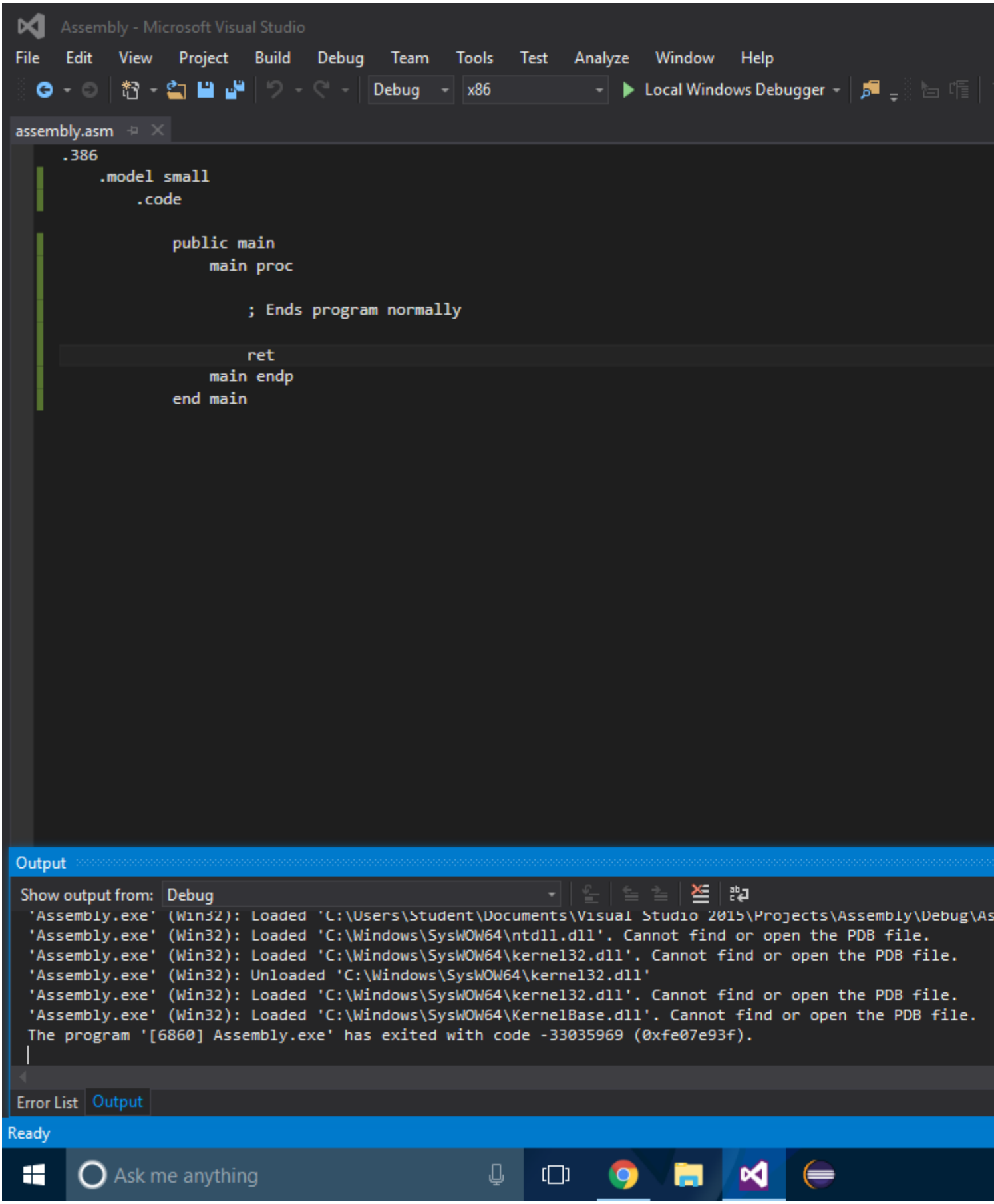
```
.386
    .model small
        .code

            public main
                main proc

                    ; Ends program normally

                    ret
                main endp
            end main
```

6 : !



: https://riptutorial.com/ko/assembly/topic/1358/--

2: glibc Linux elf64

Examples

80 % Linux, OSX Windows . ( ) . (VT100) .

.

Subrtx.asm

```
        STDIN    equ        0
       STDOUT    equ        1

     SYS_READ    equ        0
    SYS_WRITE    equ        1

    global  gets, strlen, print, atoq

            section .text
```

. .

```
; ============================================================================
; Accept canonical input from operator for a maximum of EDX bytes and replace
; terminating CR with NULL.

;        ENTER: RSI = Pointer to input buffer
;               EDX = Maximum number of characters

;        LEAVE: EAX = Number of characters entered
;               R11 = Modified by syscall, all others preserved.

;        FLAGS:  ZF = Null entry, NZ otherwise.
; ____________________________________________________________________________

     gets:  push    rcx
            push    rdi

            xor     eax, eax            ; RAX = SYS_READ
            mov     edi, eax            ; RDI = STDIN
            syscall

; TODO:    Should probably do some error trapping here, especially for
;            buffer overrun, but I'll see if it becomes an issue over time.

            dec     eax                 ; Bump back to CR
            mov     byte [rsi+rax], 0   ; Replace it with NULL

            pop     rdi
            pop     rcx
            ret
```

, write (2) . (0 - FF) . , ***strlen*** .

```
; ========================================================================
; Determine length, including terminating character EOS. Result may include
; VT100 escape sequences.

;       ENTER: RDI = Pointer to ASCII string.
;              RCX   Bits 31 - 08 = Max chars to scan (1 - 1.67e7)
;                          07 - 00 = Terminating character (0 - FF)

;       LEAVE: RAX = Pointer to next string (optional).

;       FLAGS:  ZF = Terminating character found, NZ otherwise (overrun).
; ________________________________________________________________________

   strlen:  push    rcx                 ; Preserve registers used by proc so
            push    rdi                 ; it's non-destructive except for RAX.

            mov      al, cl             ; Byte to scan for in AL.
            shr     ecx, 8              ; Shift max count into bits 23 - 00

; NOTE: Probably should check direction flag here, but I always set and
;       reset DF in the process that is using it.

            repnz   scasb               ; Scan for AL or until ECX = 0
            mov     rax, rdi            ; Return pointer to EOS + 1

            pop     rdi                 ; Original pointer for proglogue
            jz      $ + 5               ; ZF indicates EOS was found
            mov     rax, rdi            ; RAX = RDI, NULL string
            pop     rcx

            ret
```

.

```
; ========================================================================
; Display an ASCIIZ string on console that may have embedded VT100 sequences.

;       ENTER: RDI = Points to string

;       LEAVE: RAX = Number of characters displayed, including EOS
;                  = Error code if SF
;              RDI = Points to byte after EOS.
;              R11 = Modified by syscall all others preserved

;       FLAGS:  ZF = Terminating NULL was not found. NZ otherwise
;               SF = RAX is negated syscall error code.
;________________________________________________________________________

    print:  push    rsi
            push    rdx
            push    rcx

            mov     ecx, -1 << 8        ; Scan for NULL
            call    strlen
            push    rax                 ; Preserve point to next string
            sub     rax, rdi            ; EAX = End pntr - Start pntr
            jz      .done
```

```
     ; size_t = write (int STDOUT, char *, size_t length)

            mov     edx, eax            ; RDX = length
            mov     rsi, rdi            ; RSI = Pointer
            mov     eax, SYS_WRITE
            mov     edi, eax            ; RDI = STDOUT
            syscall
            or      rax, rax            ; Sets SF if syscall error

; NOTE:    This procedure is intended for console, but in the event STDOUT is
;        redirected by some means, EAX may return error code from syscall.

    .done:  pop     rdi                 ; Retrieve pointer to next string.
            pop     rcx
            pop     rdx
            pop     rsi

            ret
```

.

Generic.asm

```
global  _start

    extern  print, gets, atoq

    SYS_EXIT  equ   60
         ESC  equ   27

       BSize  equ   96

            section .rodata
   Prompt:  db  ESC, '[2J'      ; VT100 clear screen
            db  ESC, '[4;11H'   ;   "   Position cursor to line 4 column 11
            db  'ASCII -> INT64 (binary, octal, hexidecimal, decimal), '
            db  'Packed & Unpacked BCD and floating point conversions'
            db  10, 10, 0, 9, 9, 9, '=> ', 0
            db  10, 9, 'Bye'
            db  ESC, '[0m'      ; VT100 Reset console
            db  10, 10, 0

            section .text
   _start:  pop    rdi
            mov    rsi, rsp
            and    rsp, byte 0xf0       ; Align stack on 16 byte boundary.

            call   main
            mov    rdi, rax             ; Copy return code into ARG0

            mov    eax, SYS_EXIT
            syscall

; int main ( int argc, char *args[] )

     main:  enter   BSize, 0             ; Input buffer on stack
            mov     edi, Prompt
```

```
            call    print
            lea     rsi, [rbp-BSize]    ; Establish pointer to input buffer
            mov     edx, BSize          ; Max size for read(2)

    .Next:  push    rdi                 ; Save pointer to "=> "
            call    print
            call    gets
            jz      .done

            call    atoq                ; Convert string pointed to by RSI

            pop     rdi                 ; Restore pointer to prompt
            jmp     .Next

    .done:  call    print               ; RDI already points to "Bye"
            xor     eax, eax
            leave
            ret
```

Makefile

```
OBJECTS = Subrtx.o Generic.o

Generic : $(OBJECTS)
    ld -oGeneric $(OBJECTS)
    readelf -WS Generic

Generic.o : Generic.asm
     nasm -g -felf64 Generic.asm

Subrtx.o : Subrtx.asm
    nasm -g -felf64 Subrtx.asm

clean:
    rm -f $(OBJECTS) Generic
```

glibc Linux elf64 : https://riptutorial.com/ko/assembly/topic/7059/glibc---linux-elf64-

3:

?

() , . opcode ( ) ( ) .

x86 `ax` , Z80 `A` , ARM `r0` .

(RISC) , (CPU)) .

ARM (: `ADD r2,r0,r1 r2 (r0 + r1)` ,).

(x86 L0 0 100). DRAM CPU).

. / / .

Z80 "8 " 80386 "32 " . . , *Z80* 16 , 32 *80686* CPU 64 MMX .

Examples

Zilog Z80

: 8 : `A` , `B` , `C` , `D` , `E` , `H` , `L` , `F` , `I` , `R` , 16 : `SP` , `PC` , `IX` , `IY` 8b : `A'` , `B'` , `C'` , `D'` , `E'` , `H'` , `L'` `F'` .

8 `AF` , `BC` , `DE` `HL` 16 .

`SP` (`PUSH` / `POP` / `CALL` / `RET`).
`PC` .
`I` `IM 2` .
`R` CPU opcode (opcode) .
Z80 `IX` 8 `IXH:IXL` `IY` (`IYH:IYL` .

`EX AF,AF'` `AF` `AF'` `EXX` `BC,DE,HL` `BC',DE',HL'` .

:

```
    ; from other register
    LD   I,A        ; copies value in A into I (8 bit)
    LD   BC,HL      ; copies value in HL into BC (16 bit)
    ; directly with value encoded in instruction machine code
    LD   B,d8       ; 8b value d8 into B
    LD   DE,d16     ; 16b value d16 into DE
    ; from a memory (ROM/RAM)
    LD   A,(HL)     ; value from memory addressed by HL into A
    LD   A,(a16)    ; value from memory with address a16 into A
    LD   HL,(a16)   ; 16b value from memory with address a16 into HL
    POP  IX         ; 16b value popped from stack into IX
    LD   A,(IY+a8)  ; IX and IY allows addressing with 8b offset
    ; from I/O port (for writing value at I/O port use "OUT")
    IN   A,(C)      ; reads I/O port C, value goes to A
```

(: `LD H,(a16)`).

:

```
    LD   (HL),D     ; value D stored into memory addressed by HL
    LD   (a16),A    ; value A into memory with address a16
    LD   (a16),HL   ; value HL into 16b of memory with address a16
    LD   (IX+a8),d8 ; value d8 into memory at address IX+a8
    LD   (IY+a8),B  ; value B into memory at address IY+a8
    ; specials ;)
    PUSH DE         ; 16b value DE pushed to stack
    CALL a16        ; while primarily used for execution branching
      ; it also stores next instruction address into stack
```

x86

32 . 16 , 32 8 . . 16 8 32 . x86 CPU CPU

AX, BX, CX, DX, BP, SI, DI, SP 8 16 . 16 .

x86 1986 32 8 E EAX, EBX, ECX, EDX, EBP, ESI, EDI ESP.

x86_64 . RAX / r0, RBX / r3, RCX / r1, RDX / r2, RBP / r5, RSI / r6, RDI / r7, RSP / r4 64 ( ). R8, R9, R10, R11, R12, R13, R14, R15.

/ .

- AX () .
- CX () .
- DX () I / O .
- BX () (DS).
- SP () .
- BP ( ) .
- SI () (: `lodsb`) .
- DI () (: `stosb`) .

. 16 64k (16) . (: `BX` , `SI` `DI`) 64k .

6 .

SS		.
		CPU .
DS		MOV
ES		
FS		

GS		

x64

x64 x86 (x86) .

- 64 .
- 8 .
- 64 0.
- 32, 16 8 .

		()
RAX		EAX (32), AX (16), AH (8), AL (8)
RBX		EBX (32), BX (16), BH (8), BL (8)
RCX		ECX (32), CX (16), CH (8), CL (8)
RDX		EDX (32), DX (16), DH (8), DL (8)
RSI		ESI (32), SI (16), SL (8)
RDI		EDI (32), DI (16), DL (8)
RBP		EBP (32), BP (16), BPL (8)
RSP		ESP (32), SP (16), SPL (8)
R8 ~ R15		R8D-R15D (32), R8W-R15W (16), R8B-R15B (8)

.

- B , 8 ;
- W , 16 ;
- D , 32 .

: https://riptutorial.com/ko/assembly/topic/4802/

4:

. *PUSH* *POP* . . , .

Examples

Zilog Z80 Stack

`sp` ("") . `EX (sp),hl` `hl` .

"", `sp` ("") `sp` .

`sp` = `$4844` `1` , `2` , `3` (`3`). .

```
|    address | value bytes | comment (btw, all numbers are in hexadecimal)
| ---------- | ----------- | ---------------------------------
|       4840 | ?? ??       | free stack spaces to be used by next push/call
|       4842 | ?? ??       | or by interrupt call! (don't expect values to stay here)
| sp -> 4844 | 03 00       | 16 bit value "3" on top of stack
|       4846 | 02 00       | 16 bit value "2"
|       4848 | 01 00       | 16 bit value "1"
|       484A | ?? ??       | Other values in stack (up to it's origin)
|       484C | ?? ??       | like for example return address for RET instruction
```

:

```
    LD   hl,$0506
    EX   (sp),hl           ; $0003 into hl, "06 05" bytes at $4844
    POP  bc                ; like: LD c,(sp); INC sp; LD b,(sp); INC sp
                           ; so bc is now $0506, and sp is $4846
    XOR  a                 ; a = 0, sets zero and parity flags
    PUSH af                ; like: DEC sp; LD (sp),a; DEC sp; LD (sp),f
                           ; so at $4844 is $0044 (44 = z+p flags), sp is $4844
    CALL $8000             ; sp is $4842, with address of next ins at top of stack
                           ; pc = $8000 (jumping to sub-routine)
                           ; after RET will return here, the sp will be $4844 again
    LD   (L1+1),sp         ; stores current sp into LD sp,nn instruction (self modification)
    DEC  sp                ; sp is $4843
L1  LD   sp,$1234          ; restores sp to $4844 ($1234 was modified)
    POP  de                ; de = $0044, sp = $4846
    POP  ix                ; ix = $0002, sp = $4848
    ...

    ...
    ORG  $8000
    RET                    ; LD pc,(sp); INC sp; INC sp
                           ; jumps to address at top of stack, "returning" to caller
```

: `PUSH` `POP` . *LIFO* ( ) . `CALL` `JP` `CALL` . `RET` `JP` .

: (`pc`) `sp` .. `sp` " " .

: `PUSH` 11 (11t) `POP` 10t `EXX` `POP` / `PUSH` Trough `LDI` . . `PUSH` ZX Spectrum (

`DI` Interrupt).

: https://riptutorial.com/ko/assembly/topic/4957/

5:

?

. . . . !

(*INT* 80h) (keypress) ( ).

Examples

Z80 :

Z80 . . 1 . 2 . Z80 ~ 0.007s .

```
EI      ;enables Interrupts
DI      ;disables Interrupts
IM 1    ;sets the Normal Interrupt Mode


IM 2    ;sets the Advanced Interrupt Mode
LD I,$99;sets the Interrupt Pointer to $99 (just possible in IM 2)
```

: https://riptutorial.com/ko/assembly/topic/6555/

6:

. . CPU / . .

Examples

m68k IF-THEN-ELSE

```
; IF d0 == 10 GO TO ten, ELSE GO TO other
    CMP    #10,d0        ; compare register contents to immediate value 10
                         ; instruction affects the zero flag
    BEQ    ten           ; branch if zero flag set
other:
    ; do whatever needs to be done for d0 != 10
    BRA    afterother    ; unconditionally jump across IF case
ten:
    ; do whatever needs to be done for d0 == 10
afterother:
    ; continue normal common program flow
```

( ) , CPU .

FOR ... Z80

Z80 . " B 0 " DJNZ . B . FOR ... NEXT "", 0 . 8086 CPU (CPU CX) . CPU (m68k DBRA) .

```
; Trivial multiplication (by repeated adding, ignores zero in factors, so
; not recommended for general use)
;
; inputs:    A = Factor 1
;            B = Factor 2
;
; output:    A = Factor 1 * Factor 2
;
; Pseudo code
; C = A : A = 0 : FOR B = Factor 2 DOWNTO 0 : A = A + C : NEXT B

mul:
     LD    C,A        ; Save Factor 1 in C register
     XOR   A          ; Clear accumulator
mLoop:
     ADD   A,C        ; Add Factor 1 to accumulator
     DJNZ  mLoop      ; Do this Factor 2 times
     RET              ; return to caller
```

If

```
section .data
    msg_eq db 'Equal', 10
    len_eq equ $ - msg_eq
```

```
    msg_le db 'Less than', 10
    len_le equ $ - msg_le

    msg_gr db 'Greater than', 10
    len_gr equ $ - msg_gr ; Length of msg_gr
section .text
    global _main ; Make the _main label global for linker
_main:
    cmp 4, 5 ; Compare 4 and 5
    je _equal ; je = jump if equal
    jl _less ; jl = jump if less
    jg _greater ; jg = jump if greater
exit:
    ret ; Return
_equal:
    ; Whatever code here
    mov rax, 0x2000004 ; sys_write, 4 for linux
    mov rdi, 1 ; STDOUT
    mov rsi, msg_eq
    mov rdi, len_eq

    syscall

    jmp exit ; Exit
_less:
    ; Whatever code here
    mov rax, 0x2000004
    mov rdi, 1
    mov rsi, msg_le
    mov rdi, len_le

    syscall

    jmp exit
_greater:
    ; Whatever code here

    mov rax, 0x2000004
    mov rdi, 1
    mov rsi, msg_gr
    mov rdi, len_gr

    syscall
    jmp exit
```

Intel

```
section .data
    msg db 'Hello, world!', 0xA
    len equ $ - msg
section .text
global _main
_main:
    mov rax, 0 ; This will be the current number
    mov rcx, 10 ; This will be the last number

_loop:
    cmp rax, rcx
```

```
    jl .loopbody ; Jump to .loopbody if rax < rcx
    jge _exit ; Jump to _exit if rax ≥ rcx
.loopbody:
    push rax ; Store the rax value for later use

    mov rax, 0x2000004 ; 4 for Linux
    mov rdi, 1 ; STDOUT
    mov rsi, msg
    mov rdx, len

    syscall

    pop rax ; Take it back to rax

    inc rax ; Add 1 to rax. This is required since the loop must have an ending.

    jmp _loop ; Back to loop
_exit:
    ret    ; Return
```

`rax < rcx .loopbody  rax < rcx .`

: https://riptutorial.com/ko/assembly/topic/8172/-

S. No		Contributors
1		Community, Edward, FedeWar, godisgood4, old_timer, Ped7g, Pichi Wuana, sigalor, stackptr, TheFrenchPlays Hd Micraftn, Zopesconk
2	glibc Linux elf64	Shift_Left
3		FedeWar, godisgood4, Jonas W., Ped7g, Pichi Wuana, SirPython
4		Jonas W., Ped7g
5		Jonas W.
6		SpilledMango, tofro