

C AND ASSEMBLY

BLG413E – System Programming, Practice Session 1

Compiling, linking and running hello.asm

- By convention, NASM ([The Netwide Assembler](#)) source files have the **.asm** extension.
- hello.asm:

```
segment .data                ;initialized data definitions
msg db "Hello, world!",10    ;initialized data bytes (10 is ASCII code for newline)
len equ $ - msg              ;length of msg

segment .text                ;the start of a group of instructions to be assembled
global _start                ;entry label for the program

_start:
    mov eax,4                ;write system call
    mov ebx,1                ;output descriptor (standart output)
    mov ecx,msg              ;start of output buffer
    mov edx,len              ;length of output
    int 80h                  ;software interrupt 80h to implement the system call

    mov eax,1                ;exit system call
    mov ebx,0                ;return status: success
    int 80h                  ;software interrupt 80h to implement the system call
```

starting a comment

Compiling, linking and running hello.asm

- **Compilation using NASM:**

```
nasm -f elf32 hello.asm -o hello.o
```

- **Linking:**

Executable and Linkable Format (32 bit)

```
ld hello.o -o hello
```

- **Note:** If the entry label of is not `_START`, then it must be specified using the `-e` flag:

```
ld hello.o -o hello -e label
```

- **Running:**

```
./hello
```

Creating and interpreting a listing file

- A listing file (containing both the source listing of the assembly program and the hexadecimal machine code for each operation) can be created by using the -l option:

```
nasm -f elf32 hello.asm -l hello.lis -o hello.o
```

- hello.lis:

```
1                                     segment .data
2 00000000 48656C6C6F2C20776F-      msg db "Hello, world!",10
3 00000009 726C64210A
4                                     len equ $ - msg
5
6                                     segment .text
7                                     global _start
8
9 _start:
10 00000000 B804000000      mov eax,4
11 00000005 BB01000000      mov ebx,1
12 0000000A B9[00000000]    mov ecx,msg
13 0000000F BA0E000000      mov edx,len
14 00000014 CD80            int 80h
15
16 00000016 B801000000      mov eax,1
17 0000001B BB00000000      mov ebx,0
18 00000020 CD80            int 80h
```

Russian peasant method of multiplication

- Write the numbers on top of two columns.
- At each step:
 - divide the number on the 1st column by 2 (ignoring the remainder),
 - multiply the number on the 2nd column by 2,
 - stop when the number on the 1st column becomes 0.
- The result is the sum of corresponding numbers on the 2nd column with odd numbers on the 1st column.

19	22
9	44
4	88
2	176
1	352
0	
$19 \times 22 = 22 + 44 + 352$	
$= 418$	

ASM code (russian.asm) conforming to the C language calling conventions

```
1  segment .text
2  global russian
3
4  russian:
5      push ebp          ;save the old base pointer value
6      mov  ebp,esp       ;base pointer <- stack pointer
7
8      mov  ecx,[ebp+8]   ;first argument
9      mov  edx,[ebp+12]  ;second argument
10     xor   eax,eax       ;clear eax (used for returning the result)
11 next:
12     shr   ecx,1         ;divide the number on the 1st column by 2
13     jnc   even         ;even number (no carry) on the 1st column
14     add   eax,edx       ;odd number: add the 2nd column to the result
15 even:
16     shl   edx,1         ;multiply the number on the 2nd column by 2
17     cmp   ecx,0         ;stop when the number on the 1st column becomes 0
18     jne   next         ;continue if it is not 0
19
20     pop   ebp          ;restore base pointer
21     ret                ;jump to return address
```

stack layout

ebp	← esp,ebp
ret. addr.	← ebp+4
parameter	← ebp+8
parameter	← ebp+12

Usage of assembly function in a C program (rusmain.c)

- A simple C program using the russian() assembly function for multiplying two numbers.

```
1  #include <stdio.h>
2
3  int russian(int x, int y);
4
5  int main(void)
6  {
7      int x, y, z;
8
9      printf("Enter numbers: ");
10     scanf("%d %d", &x, &y);
11     z = russian(x, y);
12     printf("The product is: %d\n", z);
13     return 0;
14 }
15
```

Building the executable from russian.asm and rusmain.c

- Compile the assembly program (NASM):
`nasm -f elf32 russian.asm -o russian.o`
- Compile the C program (gcc):
`gcc -c rusmain.c -o rusmain.o`
- Link them into an executable using gcc:
`gcc russian.o rusmain.o -o russian`



Disassembling instructions in an object file

- Displays the machine instructions from an object file.
- Disassembly is done only on the sections containing instructions.

Disassembling russian.o

nasm -f elf32 russian.asm -o russian.o

objdump -d russian.o

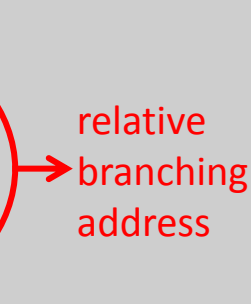
```
russian.o:      file format elf32-i386

Disassembly of section .text:

00000000 <russian>:
   0:  55                push    %ebp
   1:  89 e5             mov     %esp,%ebp
   3:  8b 4d 08          mov     0x8(%ebp),%ecx
   6:  8b 55 0c          mov     0xc(%ebp),%edx
   9:  31 c0             xor     %eax,%eax

0000000b <next>:
   b:  d1 e9             shr     %ecx
   d:  73 02             jae     11 <even>
   f:  01 d0             add     %edx,%eax

00000011 <even>:
  11:  d1 e2             shl     %edx
  13:  83 f9 00          cmp     $0x0,%ecx
  16:  75 f3             jne     b <next>
  18:  5d               pop     %ebp
  19:  c3               ret
```



relative
branching
address

Disassembling rusmain.o

gcc -c rusmain.c -o rusmain.o
objdump -d rusmain.o

As linking is not done, addresses
for printf, scanf and russian
functions are not available.

rusmain.o: file format elf32-i386

Disassembly of section .text:

00000000 <main>:

0:	55	push	%ebp
1:	89 e5	mov	%esp,%ebp
3:	83 e4 f0	and	\$0xfffffffff0,%esp
6:	83 ec 20	sub	\$0x20,%esp
9:	b8 00 00 00 00	mov	\$0x0,%eax
e:	89 04 24	mov	%eax,(%esp)
11:	e8 fc ff ff ff	printf	call 12 <main+0x12>
16:	b8 10 00 00 00	mov	\$0x10,%eax
1b:	8d 54 24 18	lea	0x18(%esp),%edx
1f:	89 54 24 08	mov	%edx,0x8(%esp)
23:	8d 54 24 14	lea	0x14(%esp),%edx
27:	89 54 24 04	mov	%edx,0x4(%esp)
2b:	89 04 24	mov	%eax,(%esp)
2e:	e8 fc ff ff ff	scanf	call 2f <main+0x2f>
33:	8b 54 24 18	mov	0x18(%esp),%edx
37:	8b 44 24 14	mov	0x14(%esp),%eax
3b:	89 54 24 04	mov	%edx,0x4(%esp)
3f:	89 04 24	mov	%eax,(%esp)
42:	e8 fc ff ff ff	russian	call 43 <main+0x43>
47:	89 44 24 1c	mov	%eax,0x1c(%esp)
4b:	b8 16 00 00 00	mov	\$0x16,%eax
50:	8b 54 24 1c	mov	0x1c(%esp),%edx
54:	89 54 24 04	mov	%edx,0x4(%esp)
58:	89 04 24	mov	%eax,(%esp)
5b:	e8 fc ff ff ff	printf	call 5c <main+0x5c>
60:	b8 00 00 00 00	mov	\$0x0,%eax
65:	c9	leave	
66:	c3	ret	

Dynamic linking of shared libraries

gcc russian.o rusmain.o -o russian.dynamic
objdump -d russian.dynamic

```
08048440 <russian>:
8048440: 55                push    %ebp
8048441: 89 e5             mov     %esp,%ebp
8048443: 8b 4d 08          mov     0x8(%ebp),%ecx
8048446: 8b 55 0c          mov     0xc(%ebp),%edx
8048449: 31 c0             xor     %eax,%eax

0804844b <next>:
804844b: d1 e9             shr     %ecx
804844d: 73 02             jae     8048451 <even>
804844f: 01 d0             add     %edx,%eax

08048451 <even>:
8048451: d1 e2             shl     %edx
8048453: 83 f9 00          cmp     $0x0,%ecx
8048456: 75 f3             jne     804844b <next>
8048458: 5d               pop     %ebp
8048459: c3               ret
```

```
0804845c <main>:
804845c: 55                push    %ebp
804845d: 89 e5             mov     %esp,%ebp
804845f: 83 e4 f0          and     $0xffffffff0,%esp
8048462: 83 ec 20          sub     $0x20,%esp
8048465: b8 a0 85 04 08    mov     $0x80485a0,%eax
804846a: 89 04 24          mov     %eax,(%esp)
804846d: e8 ce fe ff ff    call    8048340 <printf@plt>
8048472: b8 b0 85 04 08    mov     $0x80485b0,%eax
8048477: 8d 54 24 18       lea     0x18(%esp),%edx
804847b: 89 54 24 08       mov     %edx,0x8(%esp)
804847f: 8d 54 24 14       lea     0x14(%esp),%edx
8048483: 89 54 24 04       mov     %edx,0x4(%esp)
8048487: 89 04 24          mov     %eax,(%esp)
804848a: e8 e1 fe ff ff    call    8048370 <__isoc99_scanf@plt>
804848f: 8b 54 24 18       mov     0x18(%esp),%edx
8048493: 8b 44 24 14       mov     0x14(%esp),%eax
8048497: 89 54 24 04       mov     %edx,0x4(%esp)
804849b: 89 04 24          mov     %eax,(%esp)
804849e: e8 9d ff ff ff    call    8048440 <russian>
80484a3: 89 44 24 1c       mov     %eax,0x1c(%esp)
80484a7: b8 b6 85 04 08    mov     $0x80485b6,%eax
80484ac: 8b 54 24 1c       mov     0x1c(%esp),%edx
80484b0: 89 54 24 04       mov     %edx,0x4(%esp)
80484b4: 89 04 24          mov     %eax,(%esp)
80484b7: e8 84 fe ff ff    call    8048340 <printf@plt>
80484bc: b8 00 00 00 00    mov     $0x0,%eax
80484c1: c9               leave   %eax
80484c2: c3               ret
```

- Addresses for printf, scanf and russian functions are available.
- russian function is available, but printf and scanf functions are not.

Static linking of shared libraries

- `gcc -static russian.o rusmain.o -o russian.static`
- `objdump -d russian.static`
- `russian`, `printf` and `scanf` functions are available.
- File sizes of executables:
 - `russian.static` (725.5 KB) is much bigger than `russian.dynamic` (7.1 KB)
- List of shared libraries required by the program:
 - `ldd ./russian.dynamic` → `libc.so.6`
 - `ldd ./russian.static` → not a dynamic executable (shared libraries are bound during linking)

Monitoring the runtime system calls


- strace ./russian.static

```
musty@musty-VirtualBox:/media/sf_virtualbox_shared_folder/ps1/russian$ strace ./
russian.static
execve("./russian.static", ["/media/sf_virtualbox_shared_folder/ps1/russian$ strace ./
russian.static"], [/* 39 vars */]) = 0
uname({sys="Linux", node="musty-VirtualBox", ...}) = 0
brk(0)                                = 0x8217000
brk(0x8217d40)                        = 0x8217d40
set_thread_area({entry_number:-1 -> 6, base_addr:0x8217840, limit:1048575, seg_3
2bit:1, contents:0, read_exec_only:0, limit_in_pages:1, seg_not_present:0, useab
le:1}) = 0
brk(0x8238d40)                        = 0x8238d40
brk(0x8239000)                        = 0x8239000
fstat64(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 2), ...}) = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x3f
5000
fstat64(0, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 2), ...}) = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x5d
3000
write(1, "Enter numbers: ", 15Enter numbers: )          = 15
read(0, 34 15 → values are entered via keyboard
"34 15\n", 1024)          = 6
write(1, "The product is: 510\n", 20The product is: 510
)          = 20
exit_group(0)              = ?
```


Monitoring the runtime system calls

- `strace ./russian.dynamic`

open libc.so.6 → shared
libraries are bound at runtime



```
musty@musty-VirtualBox:/media/sf_virtualbox_shared_folder/ps1/russian$ strace ./
russian.dynamic
execve("./russian.dynamic", ["/russian.dynamic"], [/ * 39 vars */]) = 0
brk(0)                                = 0x84a8000
access("/etc/ld.so.nohwcap", F_OK)    = -1 ENOENT (No such file or directory)
mmap2(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x14
a000
access("/etc/ld.so.preload", R_OK)    = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat64(3, {st_mode=S_IFREG|0644, st_size=67261, ...}) = 0
mmap2(NULL, 67261, PROT_READ, MAP_PRIVATE, 3, 0) = 0x307000
close(3)                              = 0
access("/etc/ld.so.nohwcap", F_OK)    = -1 ENOENT (No such file or directory)
open("/lib/i386-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\1\0\0\0\0\0\0\0\226\1\0004\0\0\0"...,
, 512) = 512
fstat64(3, {st_mode=S_IFREG|0755, st_size=1713640, ...}) = 0
mmap2(NULL, 1723100, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x3
8b000
mmap2(0x52a000, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE
, 3, 0x19f) = 0x52a000
mmap2(0x52d000, 10972, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS
, -1, 0) = 0x52d000
close(3)                              = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x76
0000
set_thread_area({entry_number:-1 -> 6, base_addr:0x760900, limit:1048575, seg_32
bit:1, contents:0, read_exec_only:0, limit_in_pages:1, seg_not_present:0, useabl
e:1}) = 0
mprotect(0x52a000, 8192, PROT_READ)    = 0
mprotect(0x8049000, 4096, PROT_READ)   = 0
mprotect(0x63a000, 4096, PROT_READ)    = 0
munmap(0x307000, 67261)                = 0
fstat64(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 2), ...}) = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xbb
0000
fstat64(0, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 2), ...}) = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x11
0000
write(1, "Enter numbers: ", 15Enter numbers: ) = 15
read(0, 34 51
"34 51\n", 1024) = 6
brk(0)                                = 0x84a8000
brk(0x84c9000)                        = 0x84c9000
write(1, "The product is: 1734\n", 21The product is: 1734
) = 21
exit_group(0)                          = ?
```

Monitoring the library calls

```
ltrace ./russian.static
```

```
ltrace ./russian.dynamic
```

```
musty@musty-VirtualBox:/media/sf_virtualbox_shared_folder/ps1/russian$ ltrace ./
russian.static
ltrace: Couldn't find .dynsym or .dynstr in "./russian.static"
```

```
musty@musty-VirtualBox:/media/sf_virtualbox_shared_folder/ps1/russian$ ltrace ./russian.dynamic  
__libc_start_main(0x804845c, 1, 0xbfa7bd34, 0x80484d0, 0x8048540 <unfinished ...  
>  
printf("Enter numbers: ") = 15  
__isoc99_scanf(0x80485b0, 0xbfa7bc84, 0xbfa7bc88, 0x143235, 0x9d0270Enter number s: 34 15  
) = 2  
printf("The product is: %d\n", 510The product is: 510  
) = 20  
+++ exited (status 0) +++
```


Listing symbols in an object file

`nm russian.o`

`nm rusmain.o`

- t: text, T: text (global), U: undefined (imported)

```
musty@musty-VirtualBox:/media/sf_virtualbox_shared_folder/ps1/russian$ nm russian.o
00000011 t even
0000000b t next
00000000 T russian
```

```
musty@musty-VirtualBox:/media/sf_virtualbox_shared_folder/ps1/russian$ nm rusmain.o
               U __isoc99_scanf
00000000 T main
               U printf
               U russian
```

`nm /lib/libc.so.6` → `nm -D /lib/i386-linux-gnu/libc.so.6`

- printf etc...


dynamic

Debugging

- **Compiling and linking:**

nasm -f elf32 -g russian.asm

gcc -c -g rusmain.c

gcc -g russian.o rusmain.o -o russian

- **Running the debugger:**

gdb ./russian

- **Getting help:**

help (all) → get list of classes of commands

help breakpoints → breakpoints

help running → running the program

help data → examining data

help info → list of info subcommands

Debugging: Breakpoints

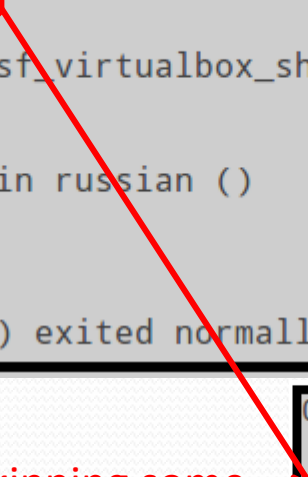
- Setting a breakpoint:
 break *function_name*
 break *line_number*
- Removing a breakpoint:
 clear *function_name*
 clear *line_number*

Debugging: Breakpoints

```
(gdb) break russian
Breakpoint 1 at 0x8048443
(gdb) run
Starting program: /media/sf_virtualbox_shared_folder/ps1/russian/russian
Enter numbers: 12 20

Breakpoint 1, 0x08048443 in russian ()
(gdb) continue
Continuing.
The product is: 240
[Inferior 1 (process 2294) exited normally]
```

skipping some
instructions !



08048440 <russian>:			
8048440:	55	push	%ebp
8048441:	89 e5	mov	%esp,%ebp
8048443:	8b 4d 08	mov	0x8(%ebp),%ecx
8048446:	8b 55 0c	mov	0xc(%ebp),%edx
8048449:	31 c0	xor	%eax,%eax
0804844b <next>:			
804844b:	d1 e9	shr	%ecx
804844d:	73 02	jae	8048451 <even>
804844f:	01 d0	add	%edx,%eax
08048451 <even>:			
8048451:	d1 e2	shl	%edx
8048453:	83 f9 00	cmp	\$0x0,%ecx
8048456:	75 f3	jne	804844b <next>
8048458:	5d	pop	%ebp
8048459:	c3	ret	

Debugging: Breakpoints

```
(gdb) clear russian
Deleted breakpoint 1
(gdb) break main
Breakpoint 2 at 0x8048465: file rusmain.c, line 9.
(gdb) run
Starting program: /media/sf_virtualbox_shared_folder/ps1/russian/russian

Breakpoint 2, main () at rusmain.c:9
9      printf("Enter numbers: ");
(gdb) continue
Continuing.
Enter numbers: 10 23
The product is: 230
[Inferior 1 (process 2297) exited normally]
```

the first executable
command in main (int x, y,
z; is skipped)

Debugging: Breakpoints

```
(gdb) clear main
Deleted breakpoint 2
(gdb) break 9
Breakpoint 3 at 0x8048465: file rusmain.c, line 9.
(gdb) run
Starting program: /media/sf_virtualbox_shared_folder/ps1/russian/russian

Breakpoint 3, main () at rusmain.c:9
9      printf("Enter numbers: ");
(gdb) continue
Continuing.
Enter numbers: 15 22
The product is: 330
[Inferior 1 (process 2299) exited normally]
```

Debugging: Running

- start : run the debugged program until the beginning of the main procedure
- run : start debugged program
- continue : continue program being debugged
- next : causes the debugger to execute the current command, stepping over function calls
- nexti : shows the next machine instruction, rather than source line (stepping over function calls)
- step : causes the debugger to execute the current command, stepping into function calls
- stepi : step by machine instructions, rather than source lines (stepping into function calls)
- *Note: "next 5", "step 5", "nexti 5" and "stepi 5" repeat same 5 times*

Debugging: Scenarios

- Scenario1:

break russian

run

//see effects of next, nexti, step and stepi

continue

- What you will observe:

- next and step have the same effect inside assembly code. They both cause the debugger to go to the next label.
- nexti and stepi have the same effect inside the assembly code. They both cause the debugger to go to the next instruction.

Debugging: Scenarios

- Scenario2:

clear russian

start // stops at beginning of main function

//see effects of next, nexti, step and stepi

- What you will observe:

- step and next behave the same.
- The debugger does not step into shared library functions and assembly functions by using step.
- stepi causes the debugger to step into shared library functions and assembly functions.

Debugging: Scenarios

- Compile and link sum.c (producing debug information using `-g` flag):
 - `gcc -g sum.c -o sum`
- Scenario3:
 - start
 - //see effects of `next`, `nexti`, `step` and `stepi`
- What you will observe:
 - `step` can be used here to step into user defined C function `sum()`.

```
1      #include <stdio.h>
2
3      int sum(int x, int y){
4          int result;
5          result = x+y;
6          return result;
7      };
8
9      int main(void)
10     {
11         int x, y, z;
12
13         printf("Enter numbers: ");
14         scanf("%d %d", &x, &y);
15         z = sum(x, y);
16         printf("The sum is: %d\n", z);
17         return 0;
18     }
```

Debugging: info

- info breakpoints:

```
(gdb) break russian
Breakpoint 1 at 0x8048443
(gdb) break 12
Breakpoint 2 at 0x80484a7: file rusmain.c, line 12.
(gdb) info breakpoints
Num      Type             Disp Enb Address      What
1        breakpoint       keep y   0x08048443  <russian+3>
2        breakpoint       keep y   0x080484a7  in main at rusmain.c:12
(gdb) clear russian
Deleted breakpoint 1
(gdb) clear 12
Deleted breakpoint 2
```

Debugging: info

- info address:

```
(gdb) info address russian
Symbol "russian" is at 0x8048440 in a file compiled without debugging.
(gdb) info address main
Symbol "main" is a function at address 0x804845c.
(gdb) info address even
Symbol "even" is at 0x8048451 in a file compiled without debugging.
(gdb) info address y
No symbol "y" in current context.
```



remember that y is not a symbol with a fixed location

Debugging: info

- info frame, backtrace and frame:

```
(gdb) break russian
Breakpoint 1 at 0x8048443
(gdb) run
Starting program: /media/sf_virtualbox_shared_folder/ps1/russian/russian
Enter numbers: 10 15

Breakpoint 1, 0x08048443 in russian ()
(gdb) info frame
Stack level 0, frame at 0xbffff2d0:
    eip = 0x8048443 in russian; saved eip 0x80484a3
    called by frame at 0xbffff300
    Arglist at 0xbffff2c8, args:
    Locals at 0xbffff2c8, Previous frame's sp is 0xbffff2d0
    Saved registers:
        ebp at 0xbffff2c8, eip at 0xbffff2cc
(gdb) backtrace
#0  0x08048443 in russian ()
#1  0x080484a3 in main () at rusmain.c:11
(gdb) frame
#0  0x08048443 in russian ()
```

Debugging: Examining registers

- info registers : list of integer registers and their contents

```
(gdb) break russian
Breakpoint 1 at 0x8048443
(gdb) run
Starting program: /media/sf_virtualbox_shared_folder/ps1/russian/russian
Enter numbers: 12 15

Breakpoint 1, 0x08048443 in russian ()
(gdb) info registers
eax                0xc          12
ecx                0x2          2
edx                0xf          15
ebx                0x2e6ff4     3043316
esp                0xbffff2c8    0xbffff2c8
ebp                0xbffff2c8    0xbffff2c8
esi                0x0          0
edi                0x0          0
eip                0x8048443     0x8048443 <russian+3>
eflags             0x286       [ PF SF IF ]
cs                 0x73        115
ss                 0x7b        123
ds                 0x7b        123
es                 0x7b        123
fs                 0x0          0
gs                 0x33        51
(gdb) info registers edx
edx                0xf          15
```

- see how the register contents change after each operation

Debugging: Examining data

- Examine memory: x/FMT ADDRESS
 - FMT is: a repeat count followed by a format letter and a size letter.
 - Format letters: o(octal), x(hex), d(decimal), u(unsigned decimal), t(binary), f(float), a(address), i(instruction), c(char) and s(string).
 - Size letters: b(byte), h(halfword), w(word), g(giant, 8 bytes).

Debugging: Examining data

- Scenario:
 - info breakpoints
 - // clear all previous breakpoints
 - break russian
 - run
 - info registers
 - // look at ebp and eip
 - // calculate ebp+8 and ebp+12
 - x/d [ebp+8]
 - x/d [ebp+12]
 - x/2d [ebp+8]
 - x/i [eip]
 - x/2i [eip]

Debugging: Examining data

```
(gdb) break russian
Breakpoint 1 at 0x8048443
(gdb) run
Starting program: /media/sf_virtualbox_shared_folder/ps1/russian/russian
Enter numbers: 12 20
Breakpoint 1, 0x08048443 in russian ()
```

```
(gdb) info registers
```

eax	0xc	12
ecx	0x2	2
edx	0x14	20
ebx	0x2e6ff4	3043316
esp	0xbffff2c8	0xbffff2c8
ebp	0xbffff2c8	0xbffff2c8
esi	0x0	0
edi	0x0	0
eip	0x8048443	0x8048443 <russian+3>
eflags	0x286	[PF SF IF]
cs	0x73	115
ss	0x7b	123
ds	0x7b	123
es	0x7b	123
fs	0x0	0
gs	0x33	51

```
(gdb) x/d 0xbffff2d0
```

```
0xbffff2d0: 12 → x/d [ebp+8]
```

```
(gdb) x/d 0xbffff2d4
```

```
0xbffff2d4: 20 → x/d [ebp+12]
```

```
(gdb) x/2d 0xbffff2d0
```

```
0xbffff2d0: 12 20 → x/2d [ebp+8]
```

```
(gdb) x/i 0x8048443
```

```
=> 0x8048443 <russian+3>: mov 0x8(%ebp),%ecx → x/i [eip]
```

```
(gdb) x/2i 0x8048443
```

```
=> 0x8048443 <russian+3>: mov 0x8(%ebp),%ecx → x/2i [eip]
0x8048446 <russian+6>: mov 0xc(%ebp),%edx
```

```
08048440 <russian>:
```

8048440:	55	push	%ebp
8048441:	89 e5	mov	%esp,%ebp
8048443:	8b 4d 08	mov	0x8(%ebp),%ecx
8048446:	8b 55 0c	mov	0xc(%ebp),%edx
8048449:	31 c0	xor	%eax,%eax

```
0804844b <next>:
```

804844b:	d1 e9	shr	%ecx
804844d:	73 02	jae	8048451 <even>
804844f:	01 d0	add	%edx,%eax

```
08048451 <even>:
```

8048451:	d1 e2	shl	%edx
8048453:	83 f9 00	cmp	\$0x0,%ecx
8048456:	75 f3	jne	804844b <next>
8048458:	5d	pop	%ebp
8048459:	c3	ret	