

The screenshot displays the OnlineGDB web interface. The main window shows the assembly code for a file named 'immediate.s'. The code includes comments and instructions for setting up a stack frame and moving immediate values into registers. The current instruction being executed is at line 15: `mov rax, 0xabcd0123456789 # 64-bit immediate`. The right sidebar shows the 'Call Stack' with one entry for 'main' at line 15. Below that, the 'Local Variables' section is empty. The 'Registers' section lists the current values for all x86-64 registers. At the bottom, the 'Debug Console' shows the user's commands: `(gdb) next`, `(gdb) next`, and `(gdb)`.

```
1 # immediate.s
2 # Some instructions to illustrate machine code.
3 .intel_syntax noprefix
4 .text
5 .globl main
6 .type main, @function
7 main:
8     push    rbp           # save caller's frame pointer
9     mov     rbp, rsp      # establish our frame pointer
10
11     mov     al, 0xab      # 8-bit immediate
12     mov     ax, 0xabcd    # 16-bit immediate
13     mov     eax, 0xabcd0123 # 32-bit immediate
14     mov     rax, 0xabcd0123456789 # to 64-bit reg
15     mov     rax, 0xabcd0123456789 # 64-bit immediate
16
17     mov     eax, 0        # return 0 to os
18     mov     rsp, rbp     # restore stack pointer
19     pop     rbp          # and frame pointer
20     ret
```

Registers:

Register	Value
rax	0xabcd0123456789
rbx	0x5555555555555555 93824992235872
rcx	0x5555555555555555 93824992235872
rdx	0x7fffffffcd8 140737488350440
rsi	0x7fffffffcd8 140737488350424
rdi	0x1 1
rbp	0x7fffffffcd8 0x7fffffffcd8
rsp	0x7fffffffcd8 0x7fffffffcd8
r8	0x0 0
r9	0x7fffffffcd8 140737354009852
r10	0x7 7
r11	0x2 2
r12	0x5555555555555555 93824992235884
r13	0x7fffffffcd8 140737488350418
r14	0x0 0
r15	0x0 0
rip	0x5555555555555555 142 <main+25>
eflags	0x246 [ PF ZF IF ]
cs	0x33 51
ss	0x2b 43
ds	0x0 0
es	0x0 0

Debug Console:

```
(gdb) next
14     mov     rax, 0xabcd0123456789 # to 64-bit reg
(gdb) next
15     mov     rax, 0xabcd0123456789 # 64-bit immediate
(gdb)
```

# OnlineGDB

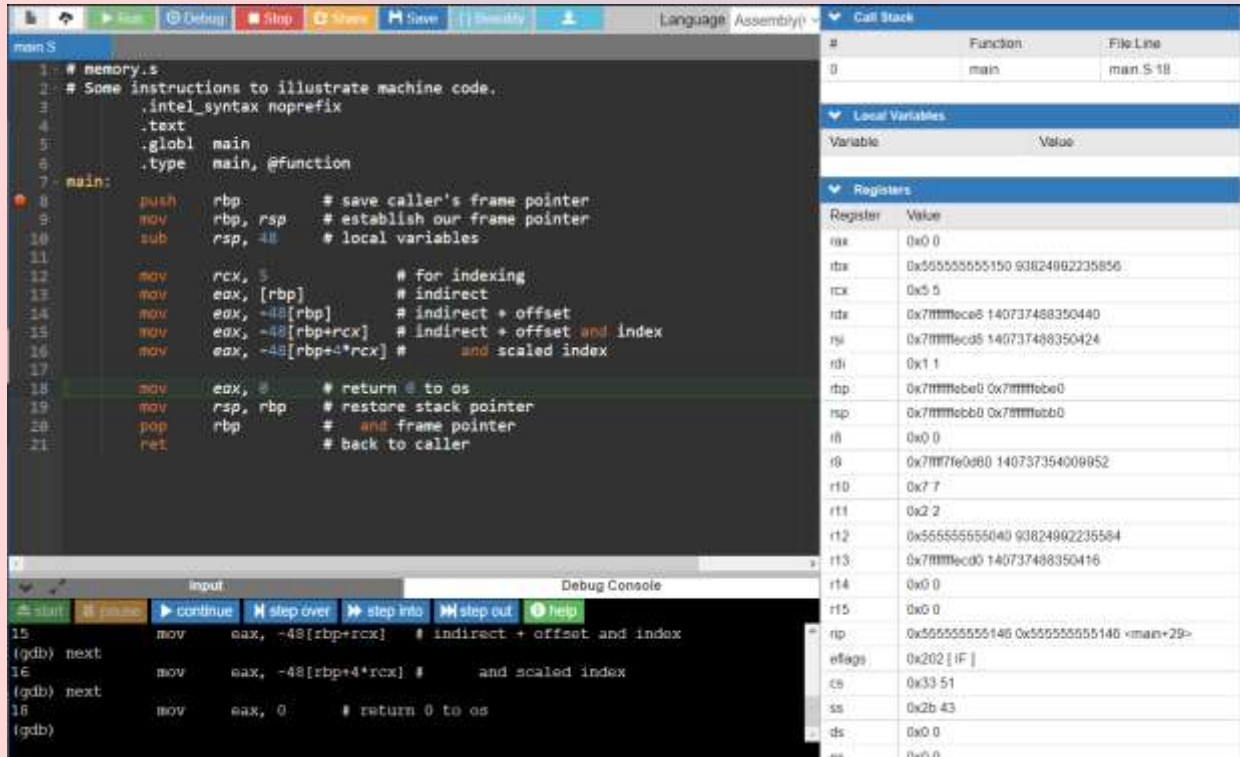
## 01-immediate.s

```
02-register.s +
1 # register.s
2 # Some instructions to illustrate machine code.
3
4 .intel_syntax noprefix
5 .text
6 .globl main
7 .type main, @function
8
9 main:
10 push rbp      # save caller's frame pointer
11 mov rbp, rsp  # establish our frame pointer
12
13 mov eax, ecx  # 32 bits, low reg codes
14 mov edi, esi  # highest reg codes
15 mov ax, cx    # 16 bits
16 mov al, cl    # 8 bits
17 mov eax, r8d  # 32 bits, 84-bit register
18 mov rax, rcx  # 64 bits
19
20 mov eax, 0    # return 0 to os
21 mov rsp, rbp  # restore stack pointer
22 pop rbp      # and frame pointer
23 ret          # back to caller
```

```
Console x Shell x +
r15 0x0 0
rip 0x40110f 0x40110f <main+13>
eflags 0x240 [ PF ZF IF ]
cs 0x33 51
ss 0x2b 43
ds 0x0 0
es 0x0 0
fs 0x0 0
gs 0x0 0
(gdb) next
16 mov rax, rcx # 64 bits
(gdb) info registers
rax 0x0 0
rbx 0x0 0
rcx 0x7fd731791598 140562224715168
rdx 0x7ffffd6131868 140736784963688
rsi 0x7ffffd6131858 140736784963672
rdi 0xd6131858 3591575648
rbp 0x7ffffd6131760 0x7ffffd6131760
rsp 0x7ffffd6131760 0x7ffffd6131760
r8 0x0 0
r9 0x7fd7317937e0 140562224723936
r10 0xfffffffffa -3846
r11 0x206 518
r12 0x401020 4198432
r13 0x0 0
r14 0x0 0
r15 0x0 0
rip 0x401112 0x401112 <main+16>
eflags 0x240 [ PF ZF IF ]
cs 0x33 51
ss 0x2b 43
ds 0x0 0
es 0x0 0
fs 0x0 0
gs 0x0 0
(gdb)
```

Repl.it

02-register.s



The screenshot displays the OnlineGDB web interface. The main window shows the assembly code for a file named `memory.s`. The code includes comments and instructions for setting up a stack frame, saving the caller's frame pointer, establishing the current frame pointer, allocating local variables, performing indexed memory access, and finally restoring the stack pointer and returning to the caller.

```
1: # memory.s
2: # Some instructions to illustrate machine code.
3: .intel_syntax noprefix
4: .text
5: .globl main
6: .type main, @function
7: main:
8:     push    rbp           # save caller's frame pointer
9:     mov     rbp, rsp      # establish our frame pointer
10:    sub     rsp, 48        # local variables
11:
12:    mov     rcx, 5         # for indexing
13:    mov     eax, [rbp]     # indirect
14:    mov     eax, -48[rbp]  # indirect + offset
15:    mov     eax, -48[rbp+rcx] # indirect + offset and index
16:    mov     eax, -48[rbp+4*rcx] # and scaled index
17:
18:    mov     eax, 0        # return 0 to os
19:    mov     rsp, rbp      # restore stack pointer
20:    pop     rbp           # and frame pointer
21:    ret
```

Below the code editor, the 'Registers' panel is visible, showing the current state of various CPU registers. The 'Call Stack' panel on the right shows the current function being executed is `main` at line 18.

Register	Value
rax	0x0 0
rbx	0x5555555555555555 93824882235856
rcx	0x5 5
rdx	0x7fffff0ce0 140737488350440
rsi	0x7fffff0cd5 140737488350424
rdi	0x1 1
rbp	0x7fffff0be0 0x7fffff0be0
rsp	0x7fffff0bb0 0x7fffff0bb0
r8	0x0 0
r9	0x7fffff0a00 140737354008852
r10	0x7 7
r11	0x2 2
r12	0x5555555555555555 93824882235854
r13	0x7fffff0cd0 140737488350416
r14	0x0 0
r15	0x0 0
rip	0x5555555555555555 146 <main+29>
eflags	0x202 [ IF ]
cs	0x33 51
ss	0x2b 43
ds	0x0 0
es	0x0 0

OnlineGDB

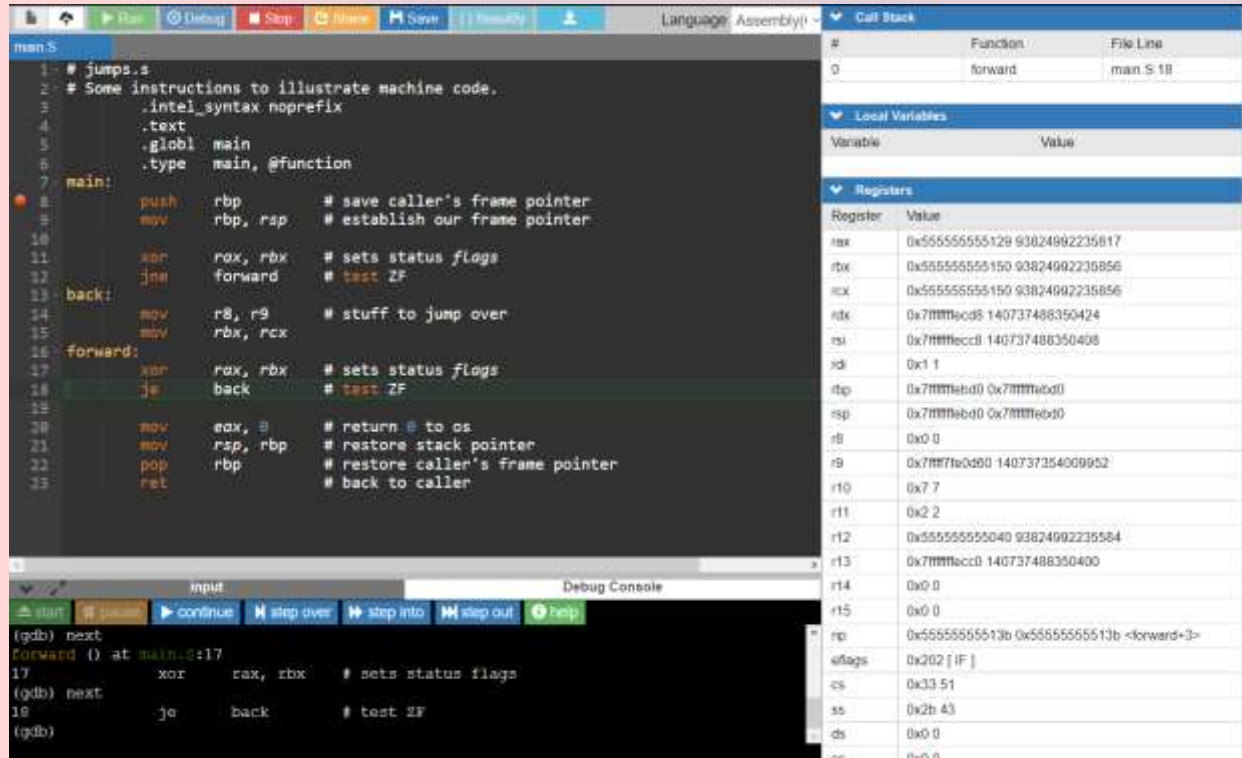
03-memory.s

```
04-constToMemory.s +
1 # constToMemory.s
2 # Some instructions to illustrate machine code.
3 .intel_syntax noprefix
4 .text
5 .globl main
6 .type main, @function
7 main:
8     push    rbp        # save caller's frame pointer.
9     mov     rbp, rsp    # establish our frame pointer
10    sub     rsp, 40      # local variables
11
12    mov     rcx, 5        # for indexing
13    mov     eax, [rbp]    # indirect
14    mov     dword ptr -48[rbp], 0x12000034 # indirect +
offset
15    mov     dword ptr 48[rbp+rcx], 0x56000078 # indirect + offset
and index
16    mov     dword ptr -48[rbp+4*rcx], 0x91000023 # and scaled
index
17
18    mov     eax, 0        # return 0 to us
19    mov     rsp, rbp      # restore stack pointer
20    pop     rbp          # and frame pointer
21    ret                # back to caller
22
```

```
3. Console x Shell x +
gs 0x0 0
(gdb) next
12 mov rcx, 5 # for indexing
(gdb)
13 mov eax, [rbp] # indirect
(gdb)
14 mov dword ptr -48[rbp], 0x12000034 # indir
ect + offset
(gdb) info registers
rax 0x401140 4198720
rbx 0x0 0
rcx 0x5 5
rdx 0x7ffcc9533c78 140723686161528
rsi 0x7ffcc9533c68 140723686161512
rdi 0x1 1
rbp 0x7ffcc9533b70 0x7ffcc9533b70
rsp 0x7ffcc9533b40 0x7ffcc9533b40
r8 0x0 0
r9 0x7fc8b8a9d7e0 140499934187616
r10 0xfffffffffffffa -3846
r11 0x206 518
r12 0x401020 4198432
r13 0x0 0
r14 0x0 0
r15 0x0 0
rip 0x401114 0x401114 <main+18>
eflags 0x202 [ IF ]
cs 0x33 51
ss 0x2b 43
ds 0x0 0
es 0x0 0
fs 0x0 0
gs 0x0 0
(gdb) next
15 mov dword ptr 48[rbp+rcx], 0x56000078 # ind
ct + offset and index
(gdb)
```

Repl.it

04-constToMemory.s



The screenshot displays the OnlineGDB interface. The main window shows the assembly code for a file named `jumps.s`. The code includes instructions for setting up a stack frame, jumping over a loop, and returning to the caller. The registers panel on the right shows the current state of the CPU registers, with the `rip` register pointing to the `forward` label.

```
1 # jumps.s
2 # Some instructions to illustrate machine code.
3 .intel_syntax noprefix
4 .text
5 .globl main
6 .type main, @function
7 main:
8     push    rbp                # save caller's frame pointer
9     mov     rbp, rsp           # establish our frame pointer
10
11     xor     rax, rbx           # sets status flags
12     jne     forward           # test ZF
13 back:
14     mov     r8, r9             # stuff to jump over
15     mov     rbx, rcx
16 forward:
17     xor     rax, rbx           # sets status flags
18     je      back              # test ZF
19
20     mov     eax, 0             # return 0 to os
21     mov     rsp, rbp           # restore stack pointer
22     pop     rbp               # restore caller's frame pointer
23     ret
```

The registers panel on the right shows the following values:

Register	Value
rax	0x55555555129 93824992235817
rbx	0x55555555150 93824992235856
rcx	0x55555555150 93824992235856
rdx	0x7fffffecc8 140737488350424
rsi	0x7fffffecc8 140737488350408
rdi	0x1 1
rbp	0x7fffffebd0 0x7fffffebd0
rsp	0x7fffffebd0 0x7fffffebd0
r8	0x0 0
r9	0x7fffffebd0 140737254009952
r10	0x7 7
r11	0x2 2
r12	0x555555555040 93824992235584
r13	0x7fffffecc0 140737488350400
r14	0x0 0
r15	0x0 0
rip	0x5555555513b 0x5555555513b <forward+3>
eflags	0x202 [ IF ]
cs	0x33 51
ss	0x2b 43
ds	0x0 0
es	0x0 0

OnlineGDB

05-jumps.s

	OnlineGDB	Repl.it	TUI GDB
Доступность	8 (требуется подключение к интернету)	8 (требуется подключение к интернету)	10 (установлена сразу в Linux)
Пользовательский интерфейс	10 (вся важная информация на экране)	8 (регистры надо вызывать самому)	6 (код отображается сразу, регистры надо вызывать самому, использование исключительно клавиатурой и стрелочками не всегда удобно)
Возможности символьной отладки на уровне исходных текстов	10	7	7
Отображение информации о регистрах	10	5	5
Простота использования	10	8	7
Наличие информации для изучения	8	8	8
Общая субъективная оценка	10	8	6

## Итоговые результаты

OnlineGDB	Repl.it	TUI GDB
66	52	49