

## 실습 2. GDB와 gdbgui

---

순천향대학교 컴퓨터공학과

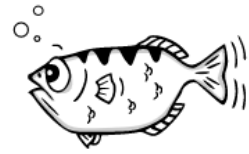
이 상 정

---

### 2.1 GDB (GNU Debugger)

## □ GDB는 GNU 디버거(debugger)

- 터미널에서 명령어로 실행파일을 디버깅
  - 프로그램의 실행
  - 행/명령어 단위 실행
  - 중단점(breakpoint) 지정
  - 변수 모니터링 및 변경
  - 레지스터 모니터링
- 유닉스 기반 시스템에서 동작
- 다양한 프로그래밍 언어 지원
- 최신 버전 8.3 (2019년 5월 기준)
- <http://www.gnu.org/software/gdb/>



## 예제 프로그램 - add.c

```
#include <stdio.h>

int main()
{
    long a, b, c;

    a = 2;
    b = 3;
    c = a + b;

    printf(" a + b -> %ld\n", c);

    return 0;
}
```

## 컴파일 및 시작

### □ 디버깅을 위해서는 **-g** 옵션으로 컴파일

- 목적파일에 **심볼의 디버깅 정보를 포함**하는 옵션

```
linux> gcc -o add -g add.c      <- 실행파일 add
linux> gdb add                  <- gdb 시작
```

```
linux> gcc -o add -g add.c
linux> gdb add
GNU gdb (Ubuntu 8.1-0ubuntu3) 8.1.0.20180409-git
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from add...done.
(gdb) □
```

## GDB 명령 - disas

### □ GDB 명령

- **disas** 함수명
  - 함수의 코드를 역 어셈블하여 보여줌
  - (gdb) disas main

```
(gdb) disas main
Dump of assembler code for function main:
0x000000000000064a <+0>:    push    %rbp
0x000000000000064b <+1>:    mov     %rsp,%rbp
0x000000000000064e <+4>:    sub     $0x20,%rsp
0x0000000000000652 <+8>:    movq    $0x2,-0x18(%rbp)
0x000000000000065a <+16>:   movq    $0x3,-0x10(%rbp)
0x0000000000000662 <+24>:   mov     -0x18(%rbp),%rdx
0x0000000000000666 <+28>:   mov     -0x10(%rbp),%rax
0x000000000000066a <+32>:   add     %rdx,%rax
0x000000000000066d <+35>:   mov     %rax,-0x8(%rbp)
0x0000000000000671 <+39>:   mov     -0x8(%rbp),%rax
0x0000000000000675 <+43>:   mov     %rax,%rsi
0x0000000000000678 <+46>:   lea     0x95(%rip),%rdi    # 0x714
0x000000000000067f <+53>:   mov     $0x0,%eax
0x0000000000000684 <+58>:   callq   0x520 <printf@plt>
0x0000000000000689 <+63>:   mov     $0x0,%eax
0x000000000000068e <+68>:   leaveq
0x000000000000068f <+69>:   retq
End of _assembler dump.
```

- 메모리
  - rbp-0x18: 2
  - rbp-0x10: 3
  - rbp-0x8 : 5
- 레지스터
  - rdx: 2
  - rax: 3
  - rax: 5

## GDB – break, run

- **break** 행번호 or 함수명

- 지정된 행(line) 또는 함수에 중단점 (breakpoint) 지정
- (gdb) break main

- **run**

- 중단점까지 실행
- (gdb) run

```
(gdb) break main
Breakpoint 1 at 0x652: file add.c, line 7.
(gdb) run
Starting program: /home/arch/ch3/add

Breakpoint 1, main () at add.c:7
7      a = 2;
(gdb) █
```

## GDB 명령 – next, step, stepi

- **next**

- 한 행을 실행 (step over), 함수 호출인 경우 함수 안으로 진입하지 않음

- **step**

- 한 행을 실행 (step into), 함수 호출인 경우 함수로 진입

- **stepi**

- 한 개의 어셈블리 명령을 실행

```
(gdb) break main
Breakpoint 1 at 0x652: file add.c, line 7.
(gdb) run
Starting program: /home/arch/ch3/add

Breakpoint 1, main () at add.c:7
7      a = 2;
(gdb) next
8      b = 3;
(gdb) next
9      c = a + b;
(gdb) next
11     printf(" a + b -> %ld\n", c);
(gdb) █
```

## – info registers, quit

```
(gdb) next
11      printf(" a + b -> %ld\n", c);
(gdb) info registers
rax          0x5          5
rbx          0x0          0
rcx          0x55555554690  93824992233104
rdx          0x2          2
rsi          0x7fffffffdf78 140737488347000
rdi          0x1          1
rbp          0x7fffffffde90 0x7fffffffde90
rsp          0x7fffffffde70 0x7fffffffde70
r8           0x7ffff7dd0d80 140737351847296
r9           0x7ffff7dd0d80 140737351847296
r10          0x0          0
r11          0x0          0
r12          0x555555554540  93824992232768
r13          0x7ffff7dd0d70 140737488346992
r14          0x0          0
r15          0x0          0
rip          0x555555554671  0x555555554671 <main+39>
eflags      0x206        [ PF IF ]
cs          0x33          51
ss          0x2b          43
ds          0x0          0
es          0x0          0
fs          0x0          0
gs          0x0          0
(gdb) next
a + b -> 5
13      return 0;
(gdb) quit
A debugging session is active.

      Inferior 1 [process 24034] will be killed.

Quit anyway? (y or n) y
linux>
```

- info registers
  - 정수 레지스 값을 표시
- quit
  - GDB 종료
- 처음부터 다시 실행 시에는 run 명령 수행

## 2.2 gdbgui

## gdbgui 소개

- ❑ GDB는 텍스트 기반 디버거로 사용이 어려움
  - `-tui` 옵션으로 텍스트 모드 GUI는 제공
  - 그래픽 기반 DDD (Data Display Debugger)를 제공하나 오래되어 한글 등 폰트 설정등 GUI가 불편
- ❑ gdbgui
  - 웹 브라우저 기반 GDB 디버거
    - 웹 브라우저 상에서 GUI 기반으로 디버깅
  - <https://www.gdbgui.com>



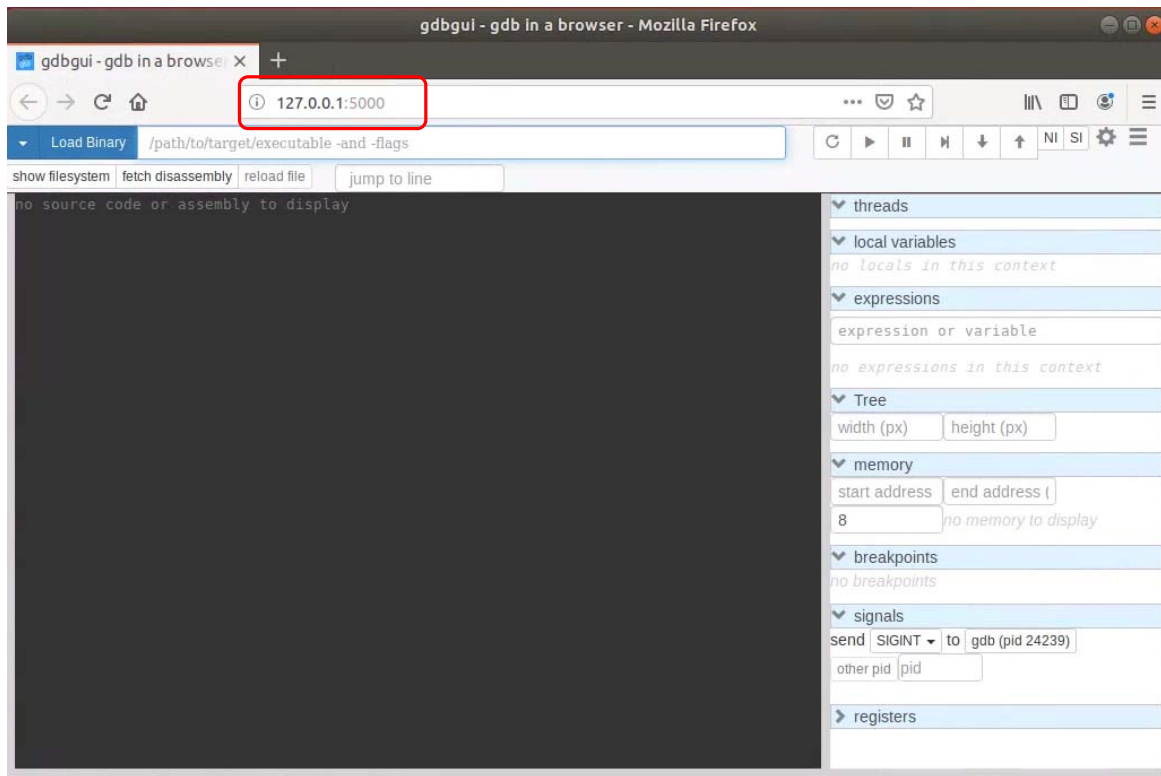
A modern, browser-based frontend to gdb (gnu debugger)

## gdbgui 설치 및 시작

- ❑ gdbgui 설치
  - pip (파이썬 패키지 설치기) 설치
    - `linux> sudo apt-get install python-pip`
  - gdbgui 설치
    - `linux> sudo pip install gdbgui`
  - gdbgui 업그레이드
    - `linux> sudo pip install --upgrade gdbgui`
- ❑ gdbgui 서버 시작
  - `linux> gdbgui`
    - gdbgui 서버를 시작하고 브라우저(Firefox)를 오픈
    - 디폴트 URL은 `http://127.0.0.1:5000`

```
linux> gdbgui
Opening gdbgui with default browser at http://127.0.0.1:5000
exit gdbgui by pressing CTRL+C
```

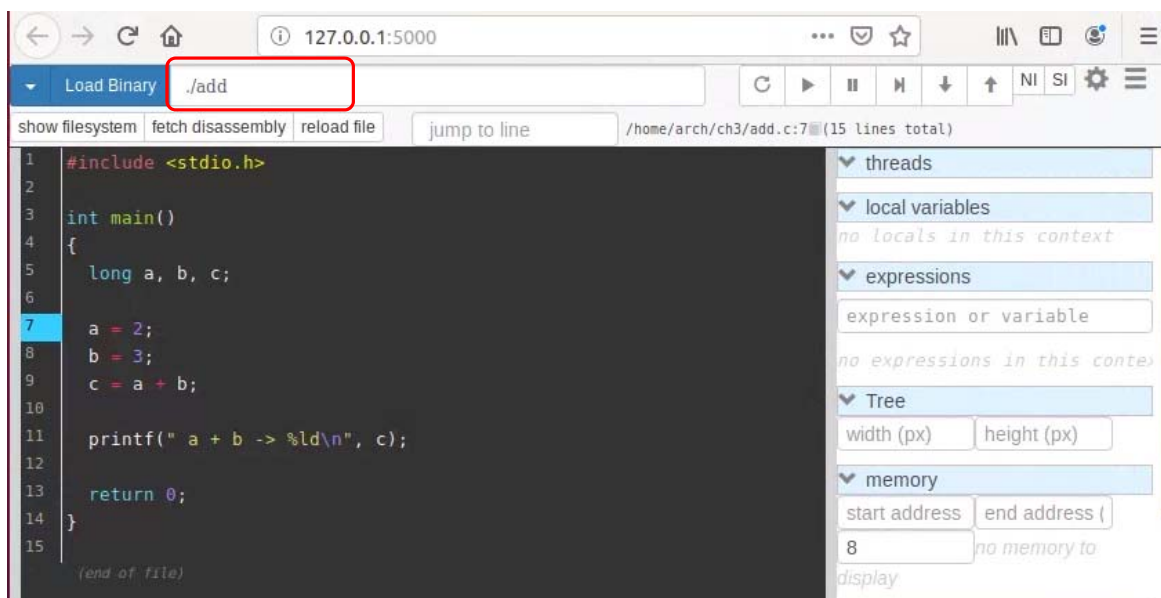
## gdbgui 브라우저



13

## 실행파일 적재

- ❑ Load Binary 입력창에 실행파일의 위치 입력
  - 실행파일이 gdbgui 실행된 디렉토리내에 있으면 **./실행파일**
  - 아니면 전체 경로 기술 **/home/lecture/실행파일**

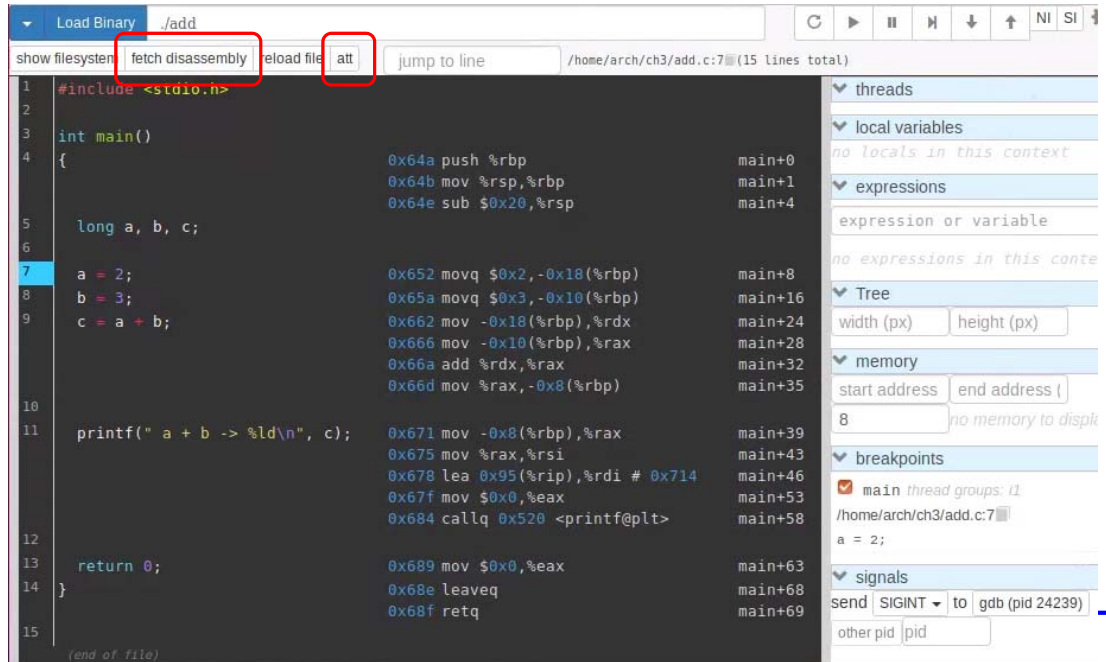


14

## 어셈블리 소스 보기 (역어셈블)

### fetch disassembly 탭을 눌러 어셈블리 소스를 보기

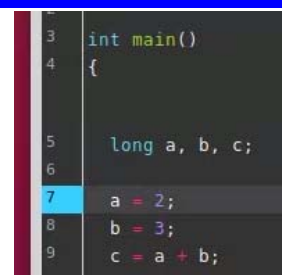
- 어셈블리 형식은 att로 설정



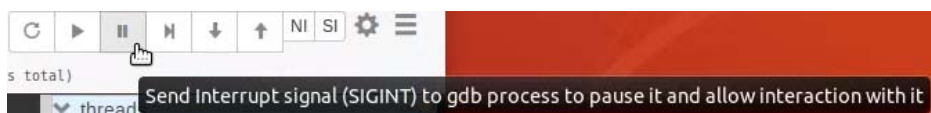
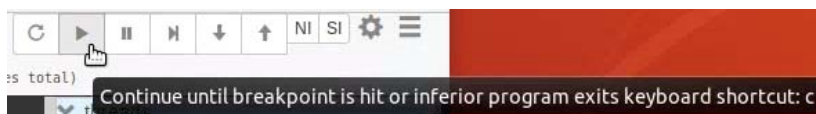
## 실행 (1)

### 오른쪽 버튼을 눌러 실행

- 초기에는 프로그램의 시작에 중단점 설정

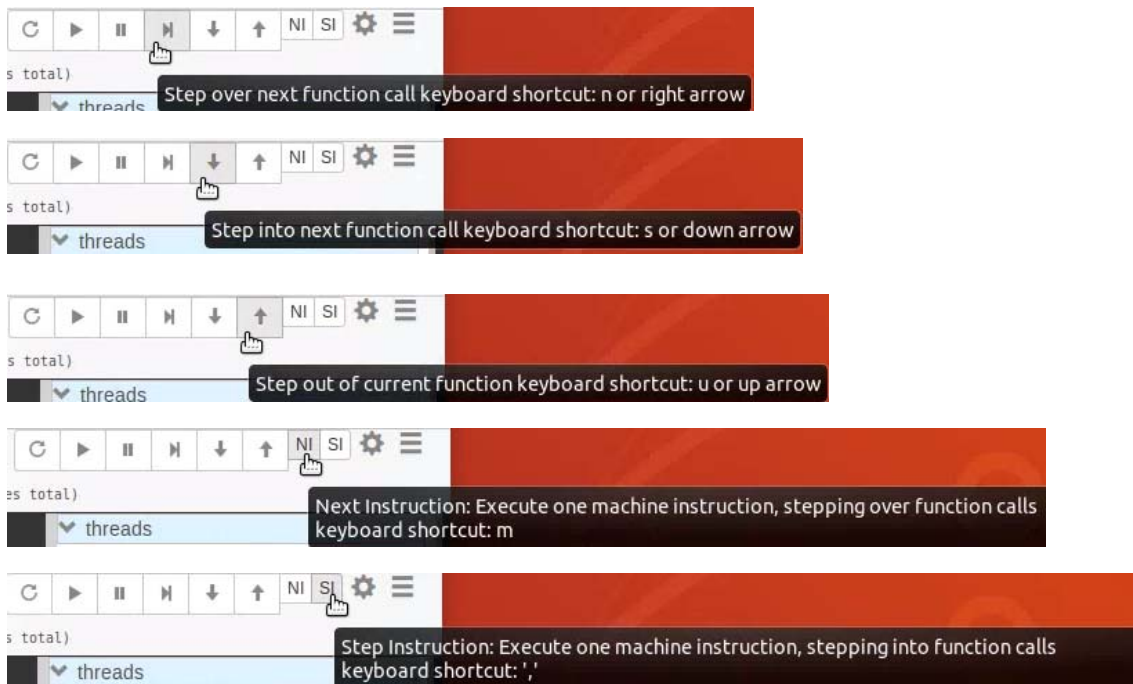


### 실행 버튼





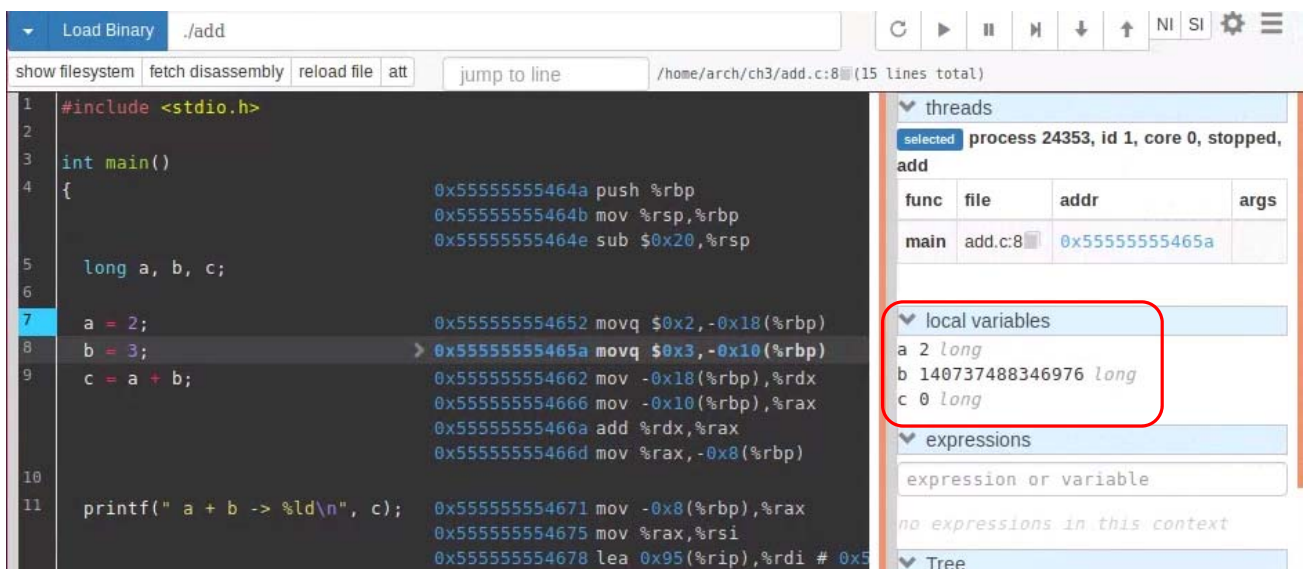


## 실행 (2)




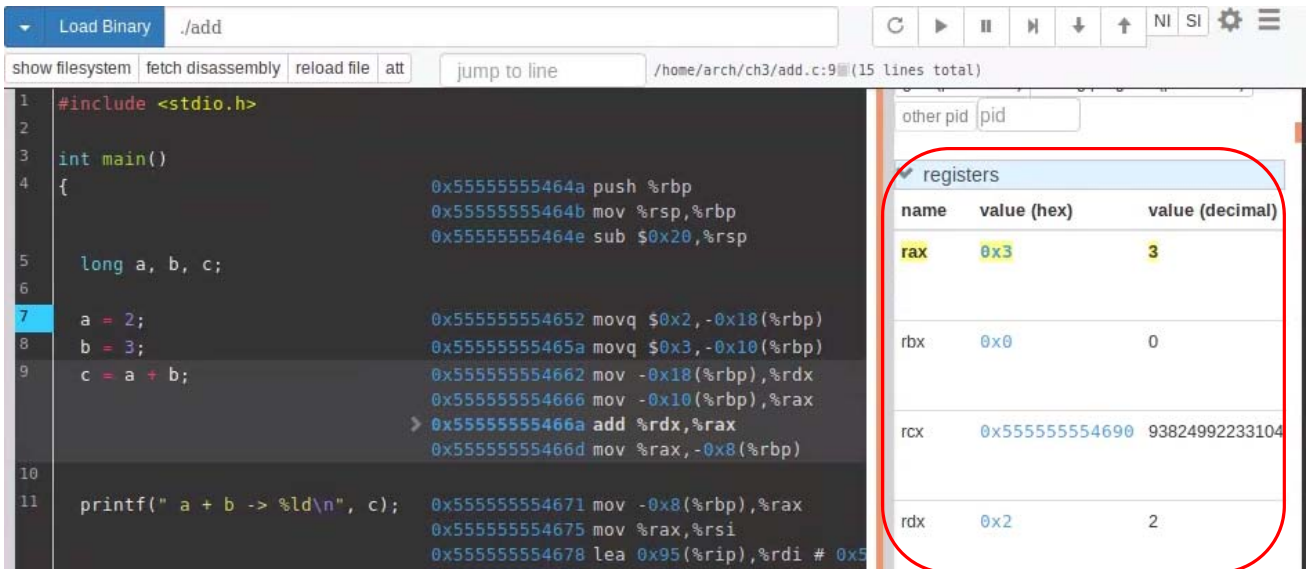
## 실행 예 - 실행 시작, 행 실행, 로컬 변수

-  으로 실행 시작
-  으로 한행 실행



## 실행 예 - 기계어 실행, 레지스터 보기

-  를 3번 눌러 3개의 기계어 명령어 실행
  - add %rdx, %rax 전까지 실행




The screenshot shows the GDB GUI with the assembly code window on the left and the registers window on the right. The assembly code is at line 9: `add %rdx, %rax`. The registers window shows the following values:

name	value (hex)	value (decimal)
rax	0x3	3
rbx	0x0	0
rcx	0x55555554690	93824992233104
rdx	0x2	2

순천대학교 컴퓨터공학과

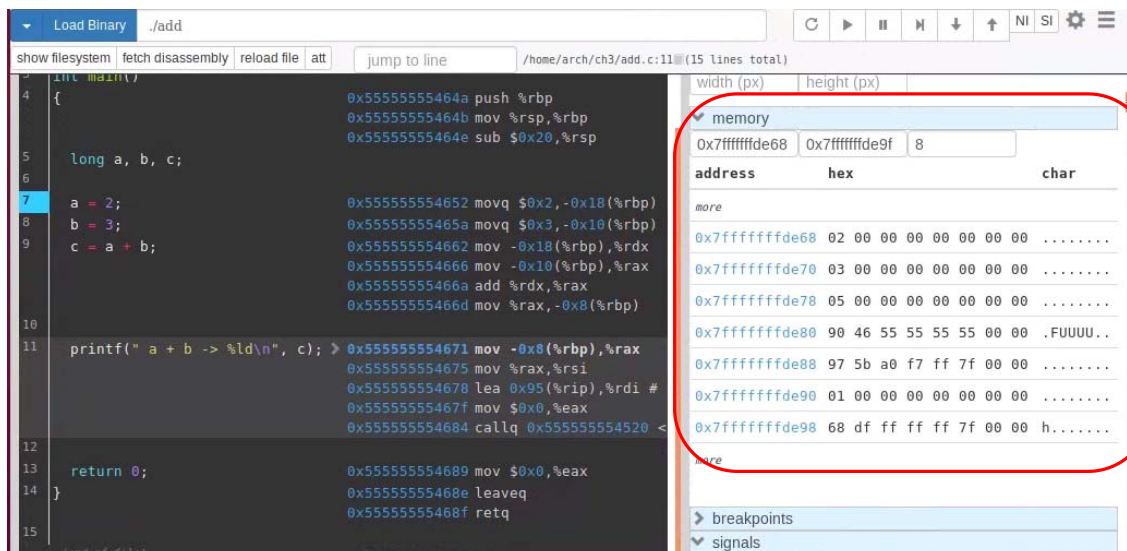
19

## 실행 예 - 메모리보기

-  으로 한 행 실행 후 메모리 보기
  - rbp 레지스터에 저장된 값을 눌러 메모리 주소 입력
  - 주변 메모리 주소에 저장된 값 보기

rbp 0x7fffffffde80 1407348834675

click to explore memory at 0x7fffffffde80



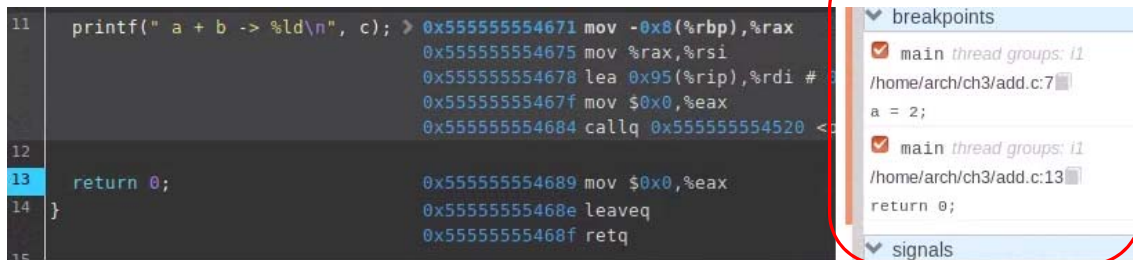
The screenshot shows the GDB GUI with the assembly code window on the left and the memory window on the right. The assembly code is at line 11: `mov -0x8(%rbp), %rax`. The memory window shows the following values:

address	hex	char
0x7fffffffde68	02 00 00 00 00 00 00 00	.....
0x7fffffffde70	03 00 00 00 00 00 00 00	.....
0x7fffffffde78	05 00 00 00 00 00 00 00	.....
0x7fffffffde80	90 46 55 55 55 55 00 00	.FUUUU..
0x7fffffffde88	97 5b a0 f7 ff 7f 00 00	.....
0x7fffffffde90	01 00 00 00 00 00 00 00	.....
0x7fffffffde98	68 df ff ff ff 7f 00 00	h.....

20

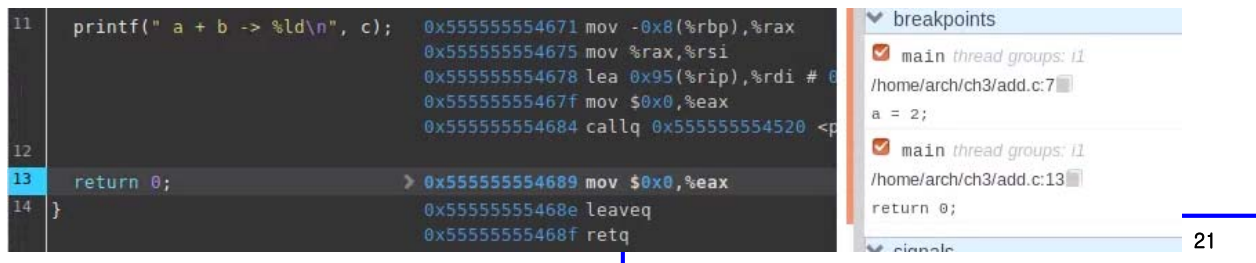
## 실행 예 - 중단점 설정

### □ 소스의 행 번호를 눌러 중단점 설정



### □ 으로 중단점까지 실행

- 프로그램 제 시적을 위해서는 실행파일 재적재



## 참고 자료

### □ GDB

- <http://www.gnu.org/software/gdb/>
- <http://beej.us/guide/bggdb/>

### □ gdbgui

- <https://www.gdbgui.com/>
- 유튜브
  - <https://www.youtube.com/channel/UCUCOSclB97r9nd54NpXMV5A>