Short Coding For Hello, world!

Baekjoon Best Conference 2024 Winter

발표자 소개

1. 티스토리 5년차 백엔드 개발자
2. 알고리즘 문제 푸는 것을 즐겼었음
3. 최근에는 리버싱에 심취함
4. 숏코딩에 특히 관심이 많음

Yun

코딩 고수가 될 거야!

algorithm # hacking # math

헬로월드 쇳코딩하는 법

73B

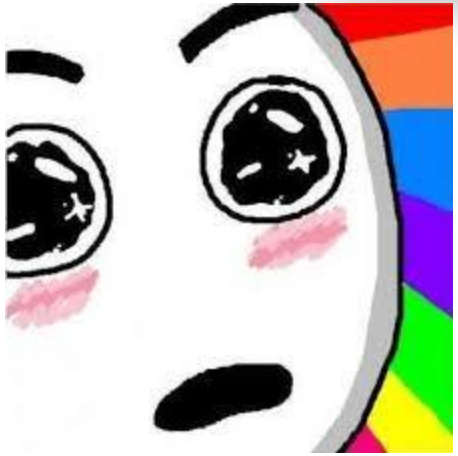
헬로월드 숏코딩하는 법

```
main(){puts("Hello, world!");}
```

30B

헬로월드 쇳코딩하는 법

```
main(){puts("Hello, world!");}
```



30B

Thank you for watching

슬라이드 쇼가 끝났습니다. 끝내려면 마우스를 클릭하십시오.

Short Coding For Hello, world!

Baekjoon Best Conference 2024 Winter

Short Coding Fo

Hello, world!

Baekjoon Best Conference 2024 Winter

In BINARY

조건 및 목표

0. 환경 : Ubuntu 24.04 에서 진행

1. x86-64(AMD64/intel64) 리눅스에서 정상 실행되어야 함

2. 바이너리의 크기가 작을수록 좋음

3. "Hello, world!" 출력 후 개행까지 할 것

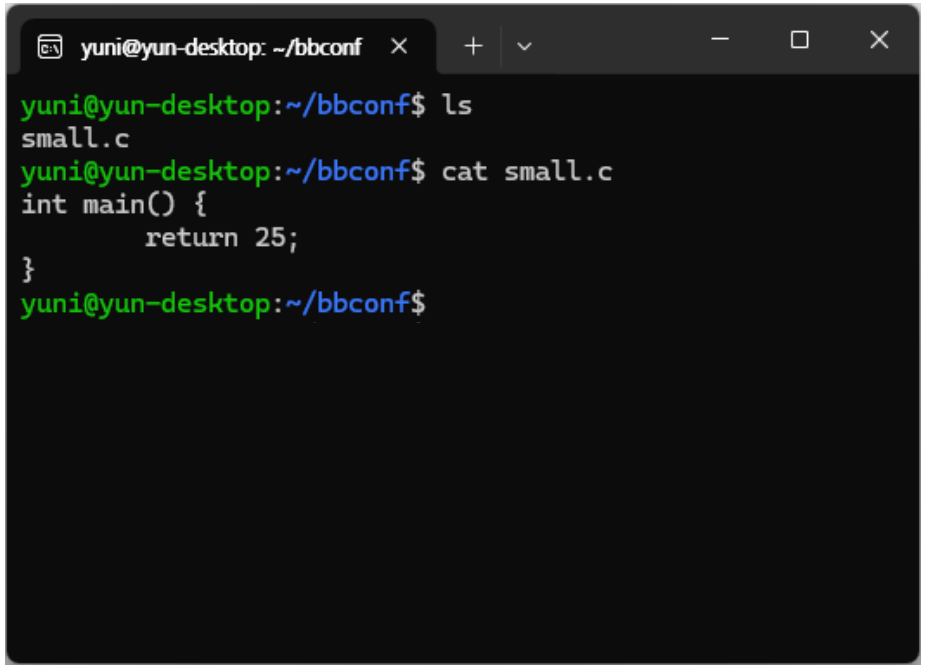
4. OS에 0을 반환하고 정상 종료

가장 간단한 프로그램의 크기

```
int main() {
    return 25;
}
```

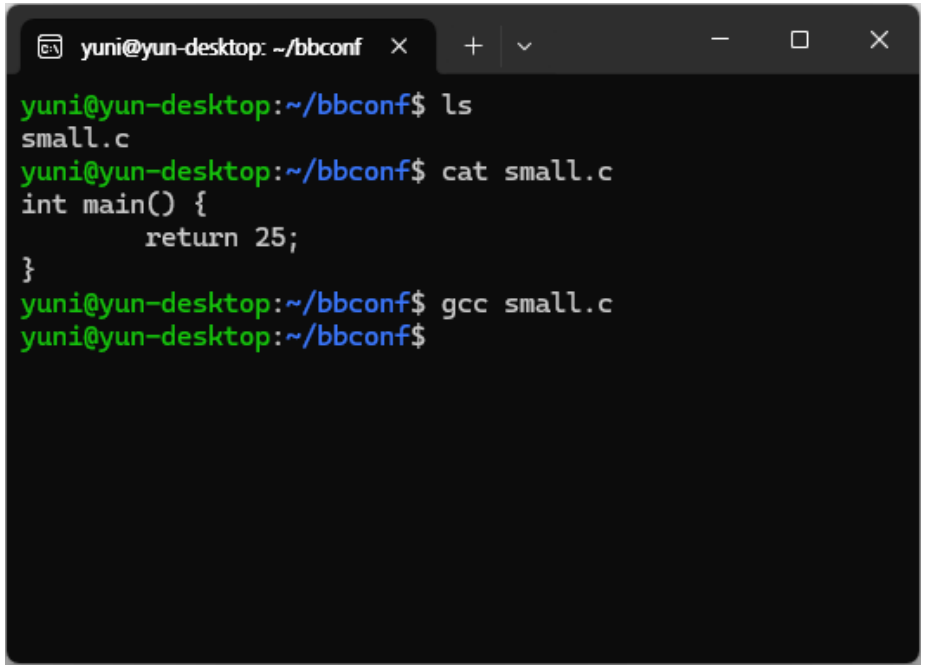
가장 간단한 프로그램의 크기

```c
int main() {
    return 25;
}
```



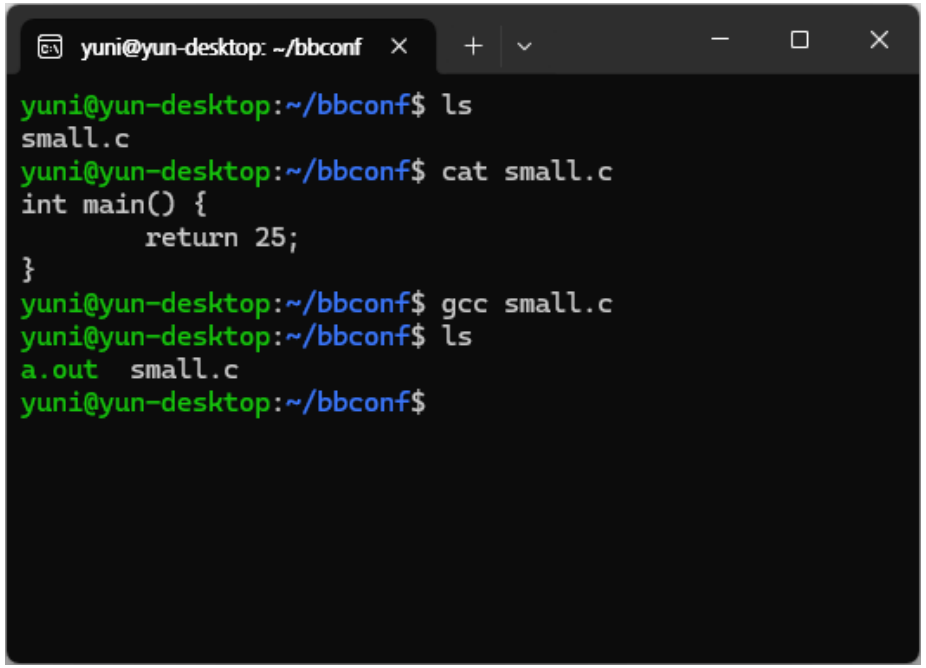
가장 간단한 프로그램의 크기

```c
int main() {
    return 25;
}
```



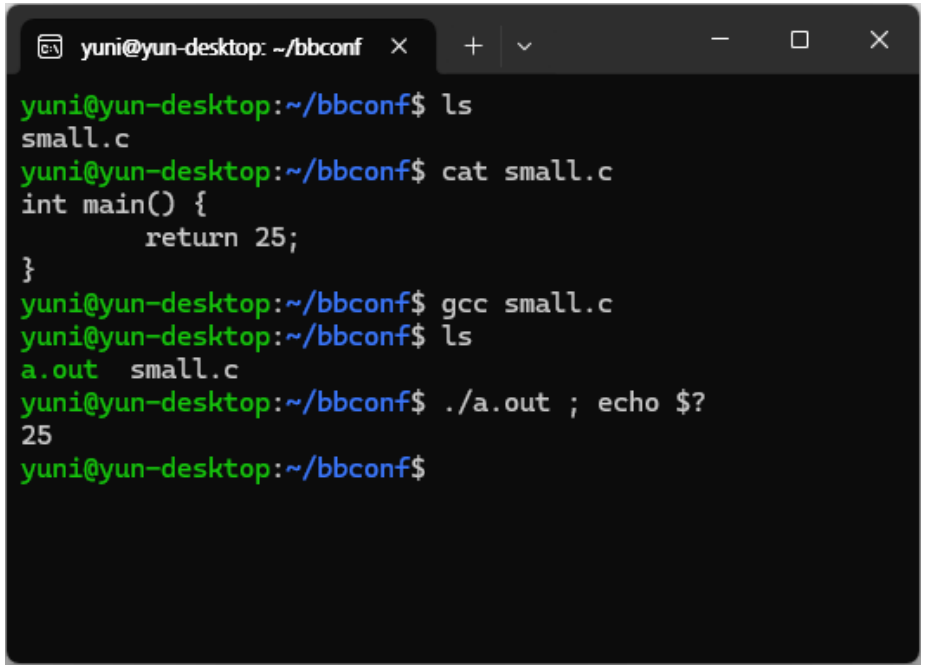
가장 간단한 프로그램의 크기

```c
int main() {
    return 25;
}
```



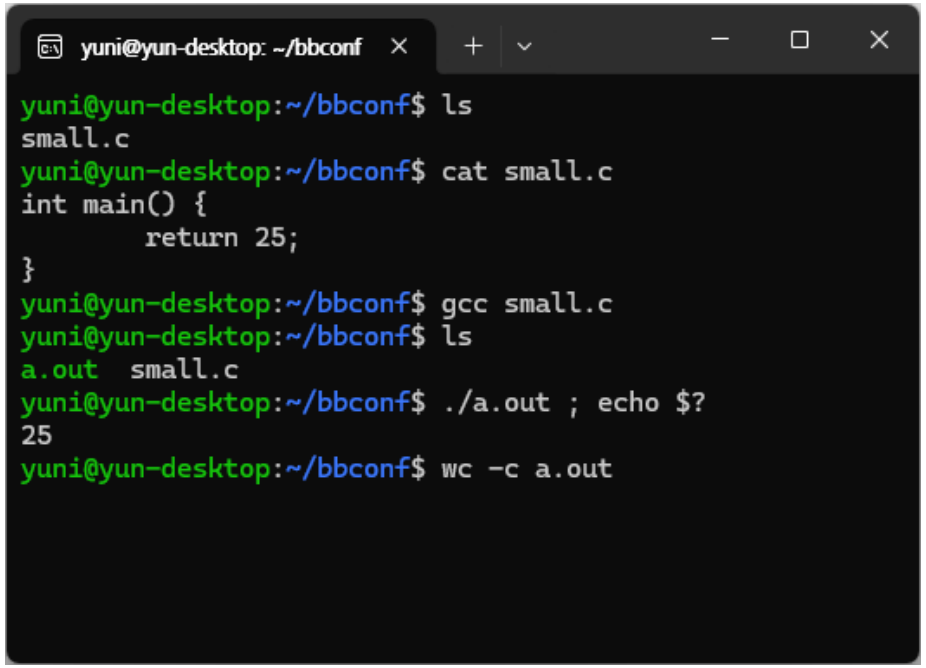
가장 간단한 프로그램의 크기

```c
int main() {
    return 25;
}
```



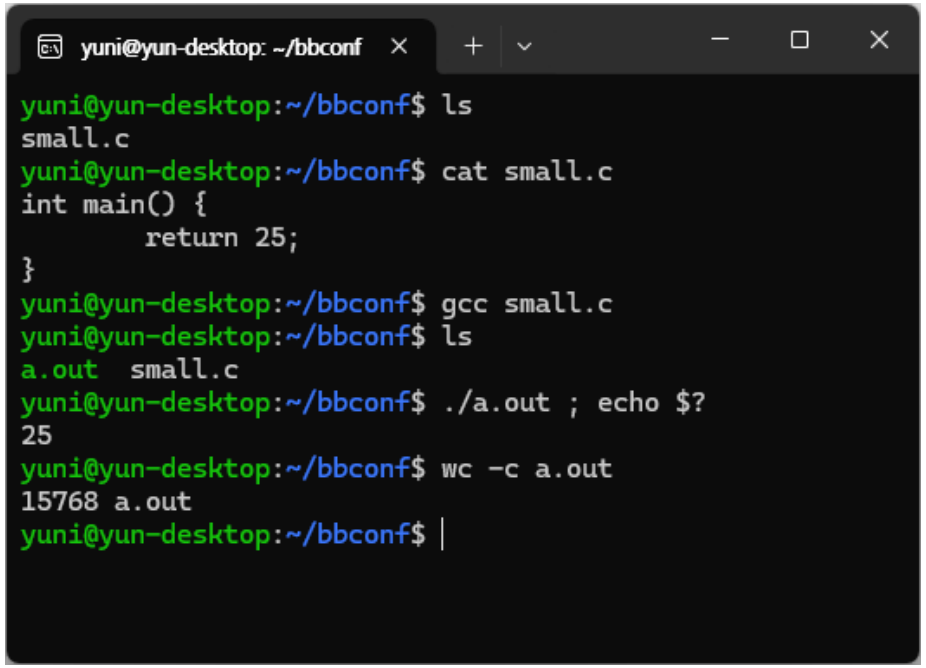
가장 간단한 프로그램의 크기

```c
int main() {
    return 25;
}
```



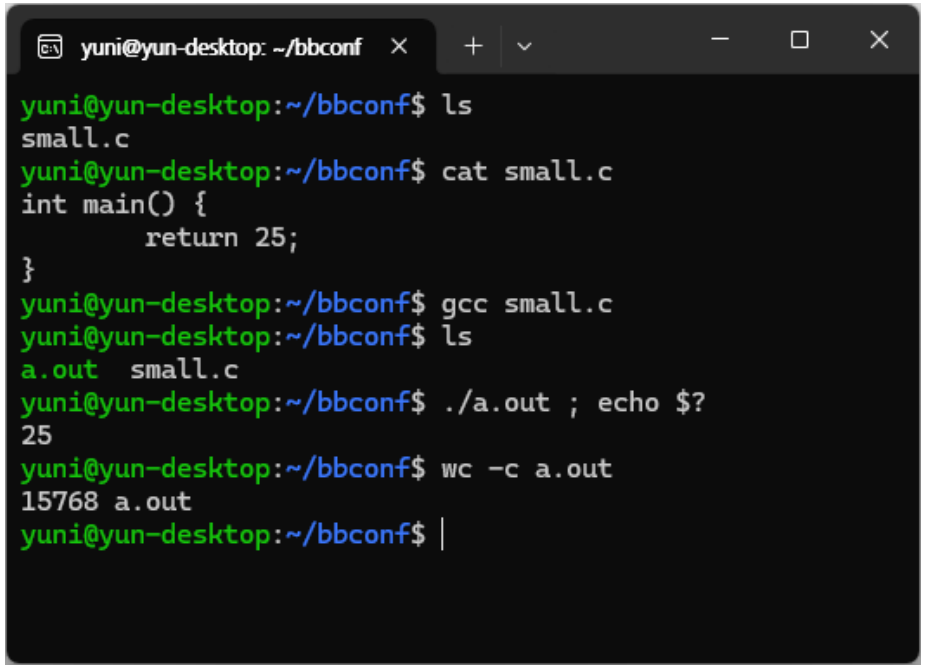
가장 간단한 프로그램의 크기

```c
int main() {
    return 25;
}
```



가장 간단한 프로그램의 크기

```c
int main() {
    return 25;
}
```



Best record **15768** Bytes

gcc의 -s 옵션

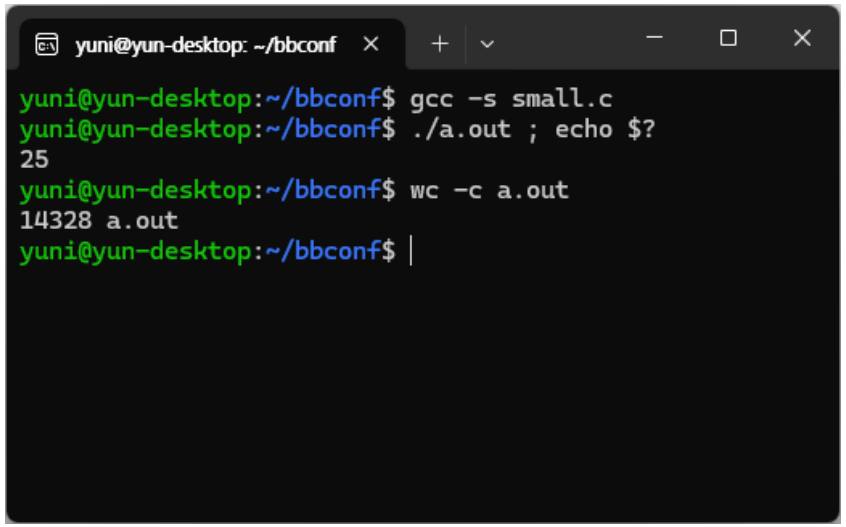
-rdynamic
Pass the flag **-export-dynamic** to the ELF linker, on targets that support it. This instructs the linker to add all symbols, not only used ones, to the dynamic symbol table. This option is needed for some uses of "dlopen" or to allow obtaining backtraces from within a program.

-s Remove all symbol table and relocation information from the executable.

-static
On systems that support dynamic linking, this overrides **-pie** and prevents linking with the shared libraries. On other systems, this option has no effect.

Best record **15768** Bytes

gcc의 -s 옵션



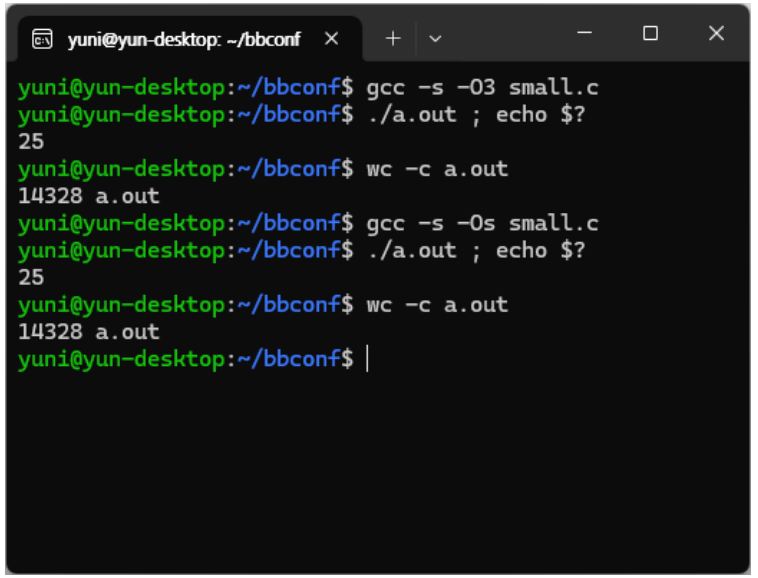
Best record **15768** -> **14328** Bytes (▼1440)

gcc 최적화 옵션

```
-O3 Optimize yet more.  -O3 turns on all optimizations specified
    by -O2 and also turns on the following optimization flags:

    -fgcse-after-reload -finline-functions -fipa-cp-clone
    -floop-interchange -floop-unroll-and-jam -fpeel-loops
    -fpredictive-commoning -fsplit-paths
    -ftree-loop-distribute-patterns -ftree-loop-distribution
    -ftree-loop-vectorize -ftree-partial-pre -ftree-slp-vectorize
    -funswitch-loops -fvect-cost-model
    -fversion-loops-for-strides

-Os Optimize for size.  -Os enables all -O2 optimizations except
    those that often increase code size:

    -falign-functions  -falign-jumps -falign-labels
    -falign-loops -fprefetch-loop-arrays
    -freorder-blocks-algorithm=stc

    It also enables -finline-functions, causes the compiler to
    tune for code size rather than execution speed, and performs
    further optimizations designed to reduce code size.
```

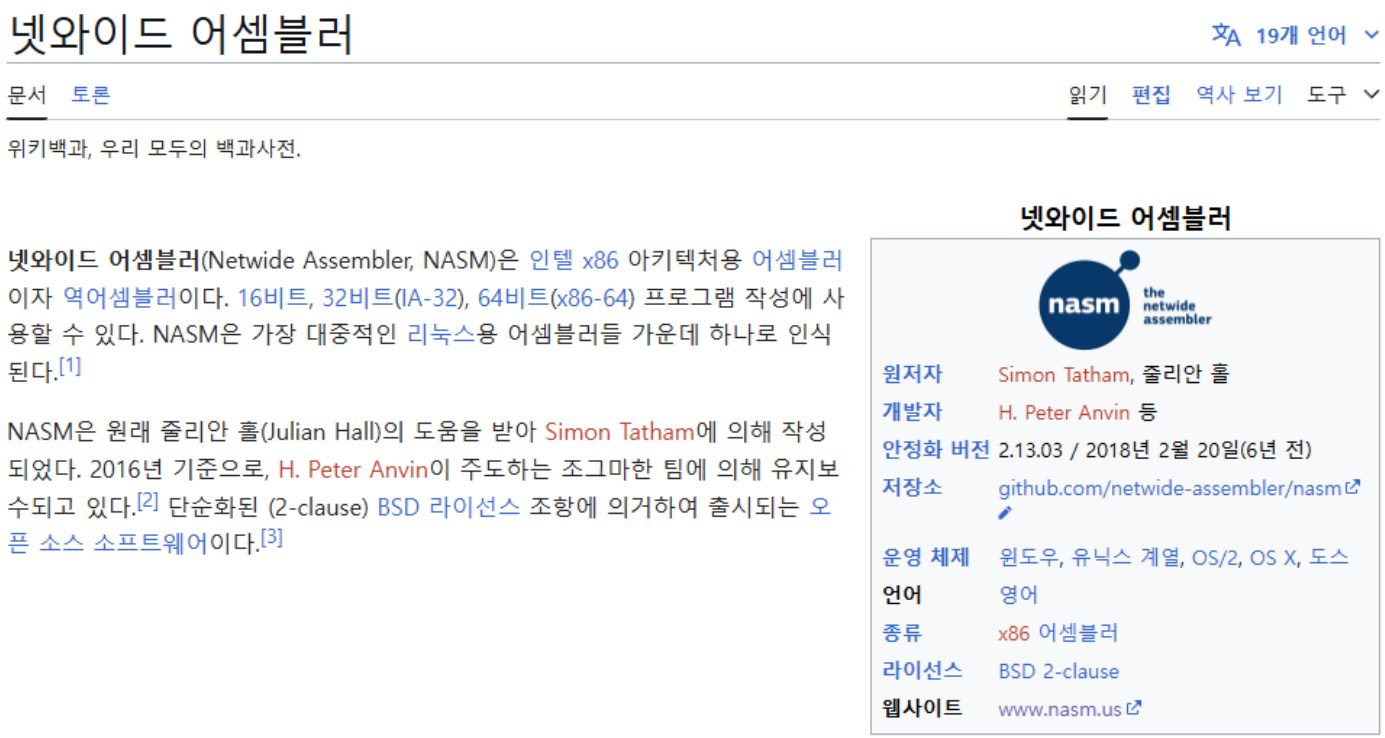
Best record **14328** Bytes

gcc 최적화 옵션



Best record **14328** Bytes

nasm으로 도전



넷와이드 어셈블러

19개 언어

문서 토론 읽기 편집 역사 보기 도구

위키백과, 우리 모두의 백과사전.

넷와이드 어셈블러(Netwide Assembler, NASM)은 인텔 x86 아키텍처용 어셈블러이자 역어셈블러이다. 16비트, 32비트(IA-32), 64비트(x86-64) 프로그램 작성에 사용할 수 있다. NASM은 가장 대중적인 리눅스용 어셈블러들 가운데 하나로 인식된다.[1]

NASM은 원래 줄리안 홀(Julian Hall)의 도움을 받아 Simon Tatham에 의해 작성되었다. 2016년 기준으로, H. Peter Anvin이 주도하는 조그마한 팀에 의해 유지보수되고 있다.[2] 단순화된 (2-clause) BSD 라이선스 조항에 의거하여 출시되는 오픈 소스 소프트웨어이다.[3]

넷와이드 어셈블러

nasm the netwide assembler

원저자	Simon Tatham, 줄리안 홀
개발자	H. Peter Anvin 등
안정화 버전	2.13.03 / 2018년 2월 20일(6년 전)
저장소	github.com/netwide-assembler/nasm
운영 체제	윈도우, 유닉스 계열, OS/2, OS X, 도스
언어	영어
종류	x86 어셈블러
라이선스	BSD 2-clause
웹사이트	www.nasm.us

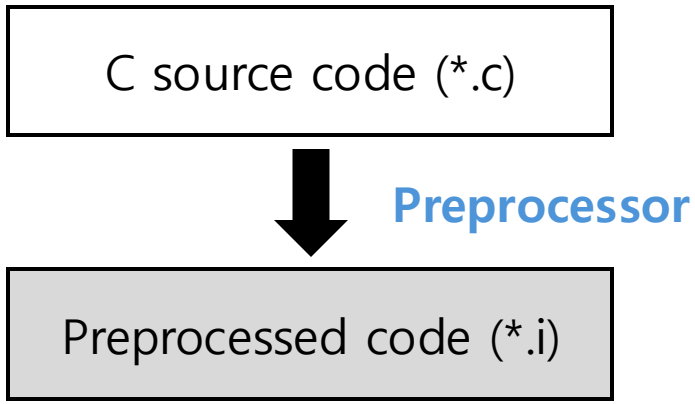
Best record **14328** Bytes

nasm으로 도전

C source code (*.c)

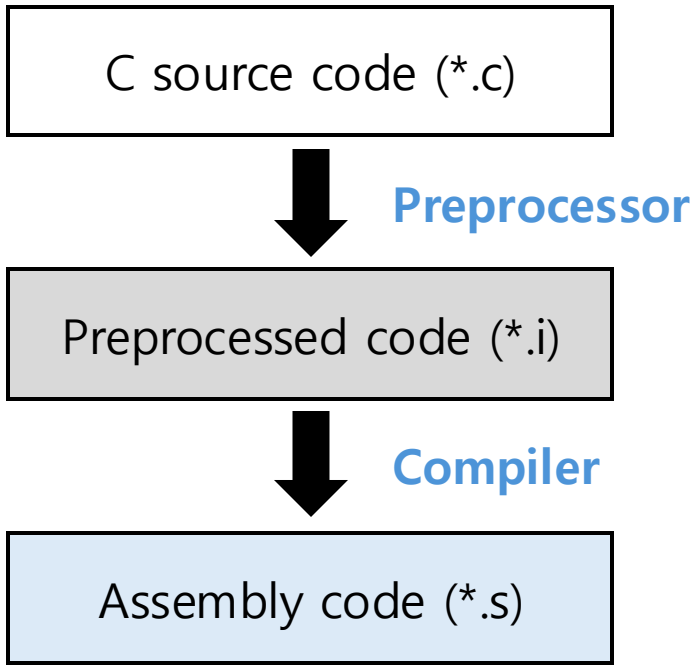
Best record **14328** Bytes

nasm으로 도전



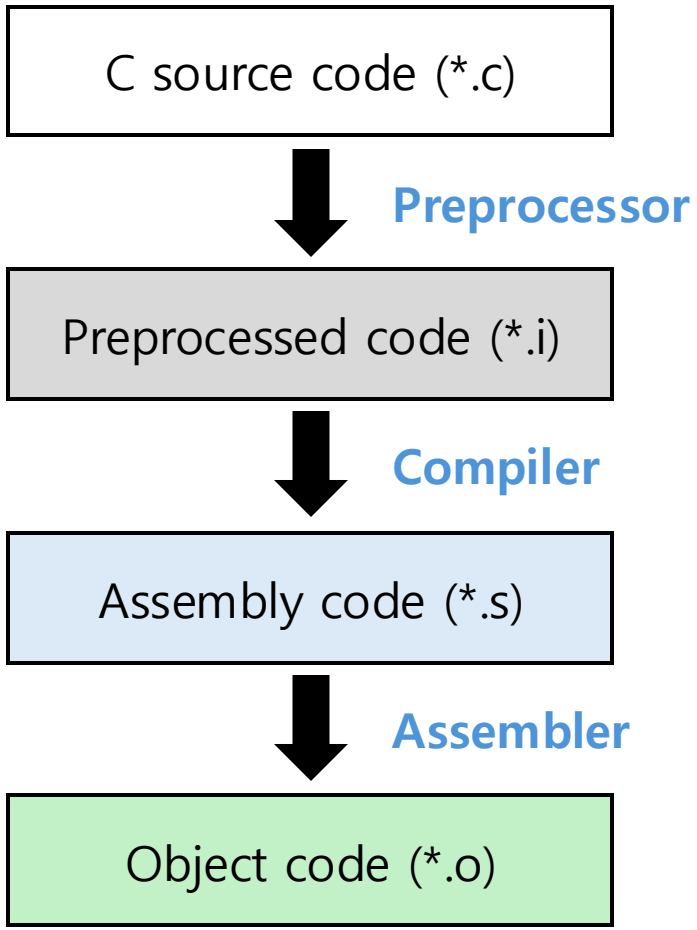
Best record **14328** Bytes

nasm으로 도전



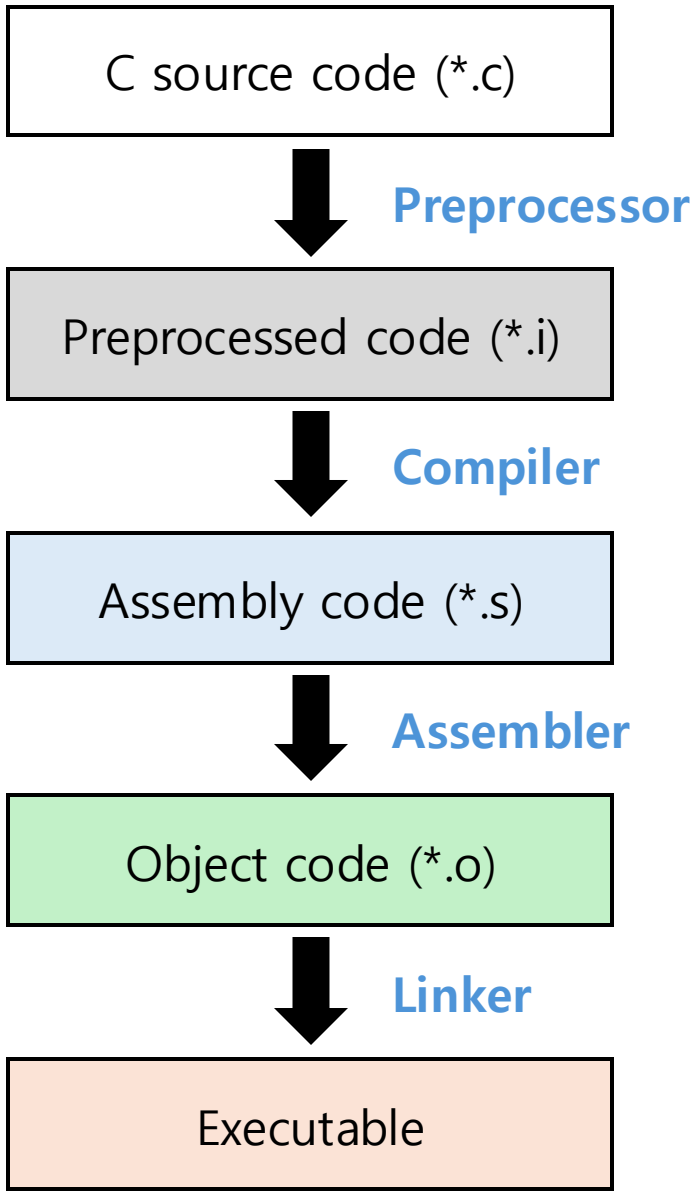
Best record **14328** Bytes

nasm으로 도전



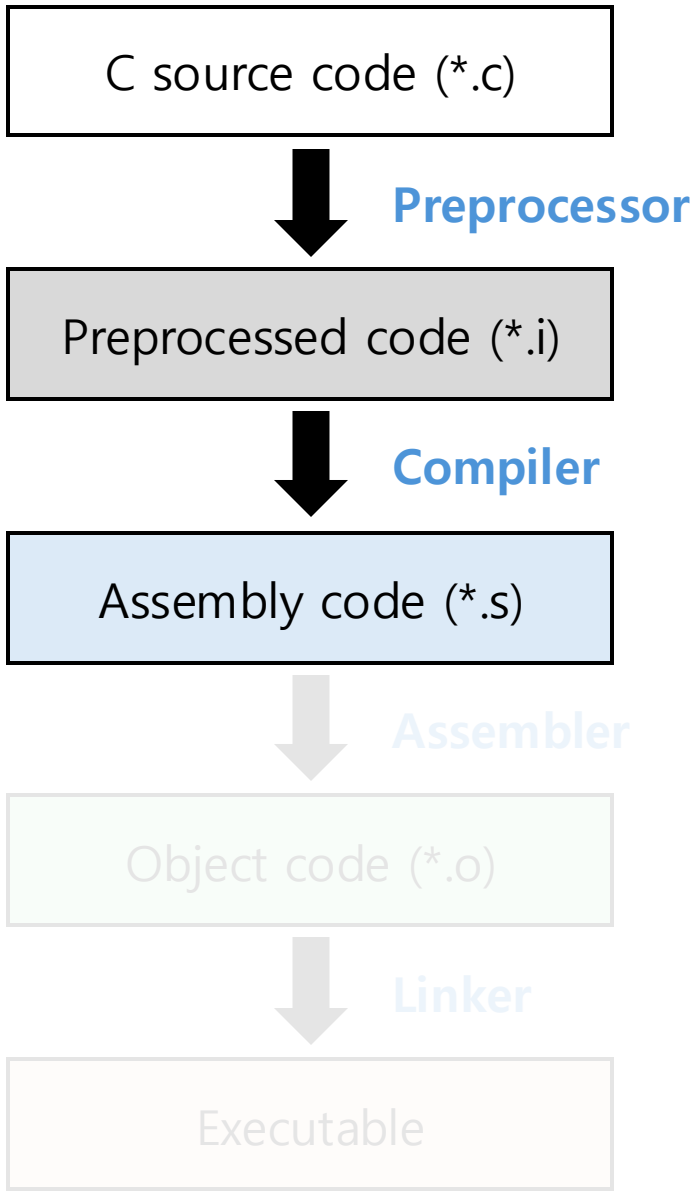
Best record **14328** Bytes

nasm으로 도전



Best record **14328** Bytes

nasm으로 도전



Best record **14328** Bytes

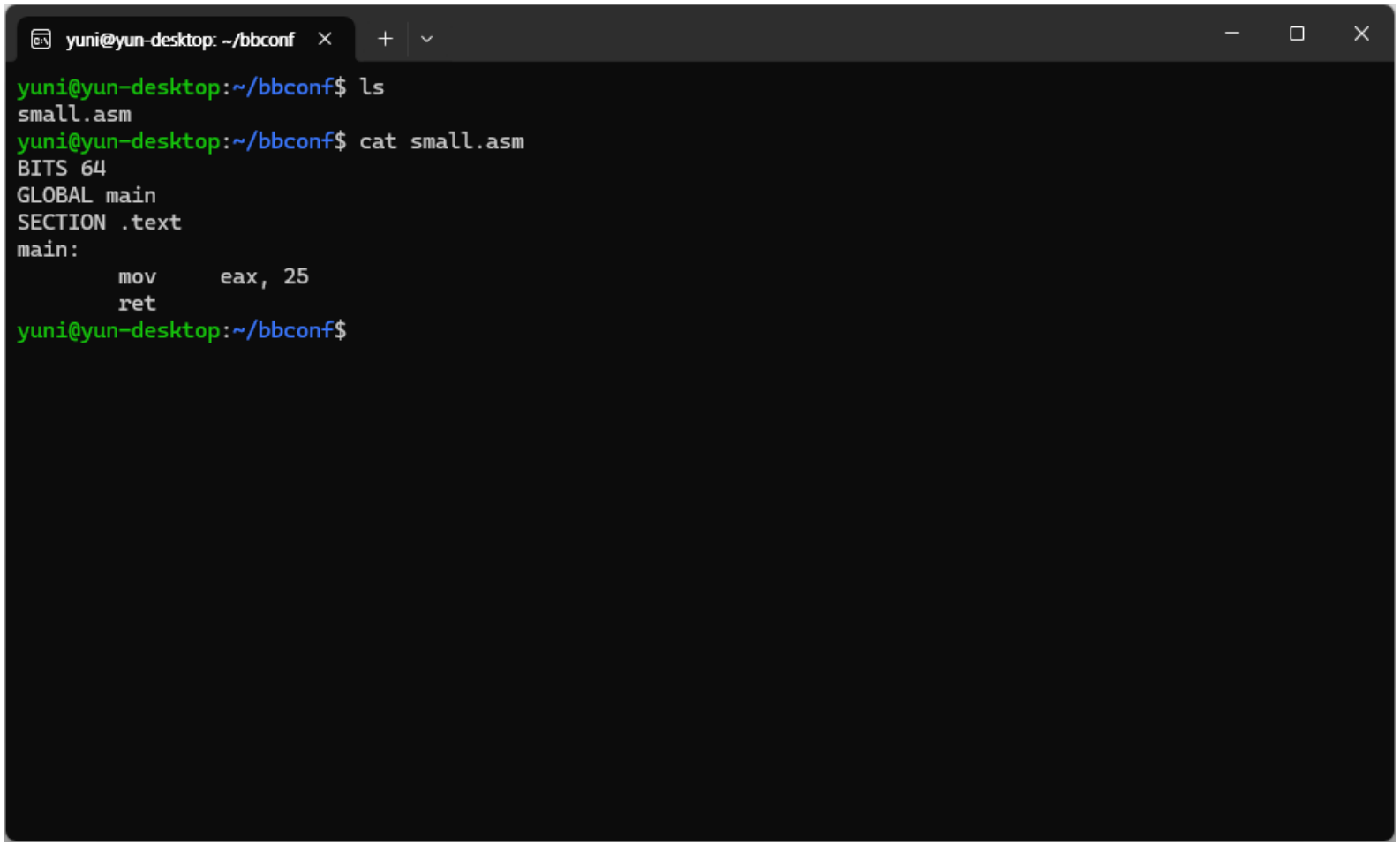
nasm으로 도전

```
BITS 64
GLOBAL main
SECTION .text
main:
    mov     eax, 25
    ret
```

Best record **14328** Bytes

nasm으로 도전

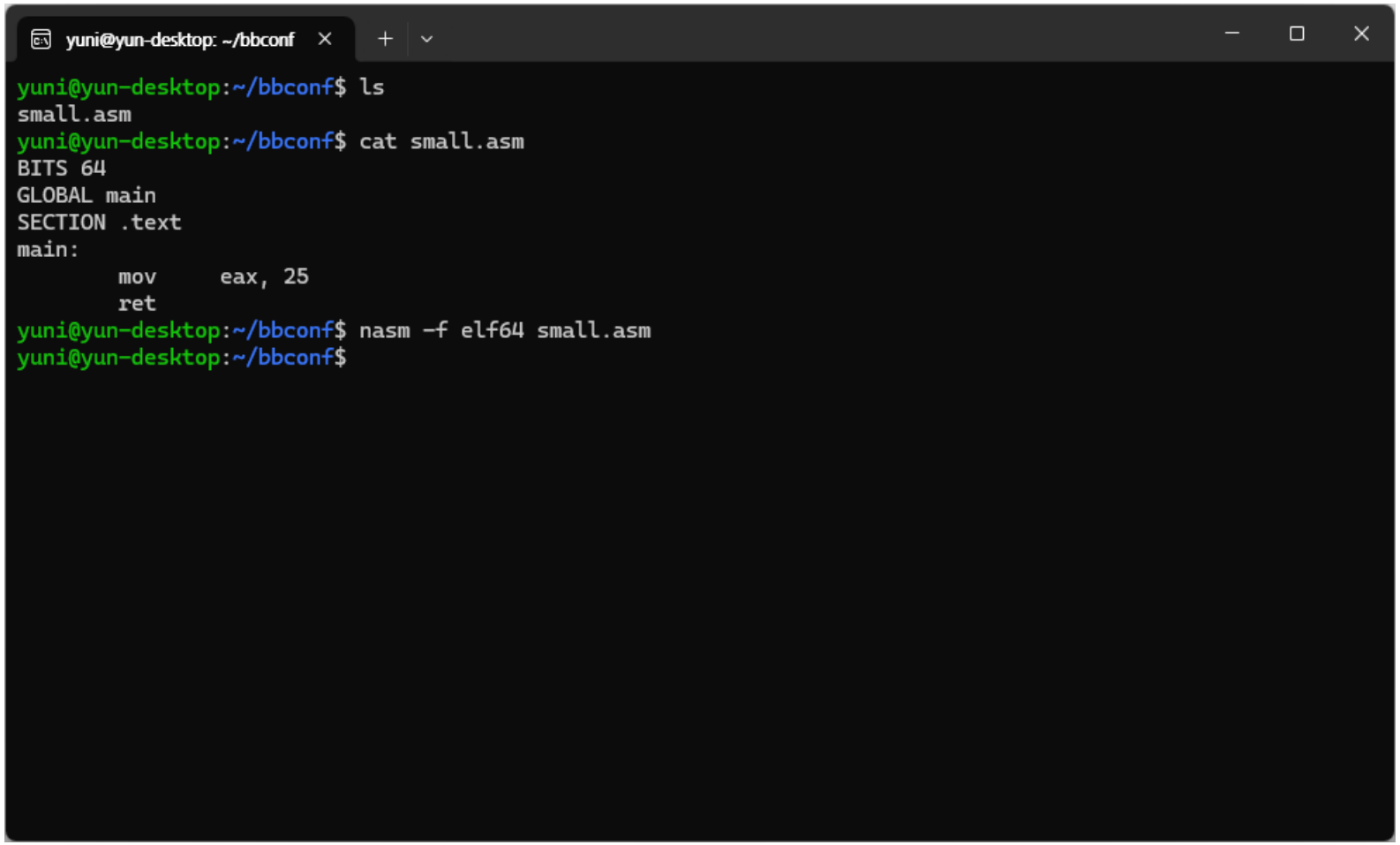
```
BITS 64
GLOBAL main
SECTION .text
main:
    mov     eax, 25
    ret
```



Best record **14328** Bytes

nasm으로 도전

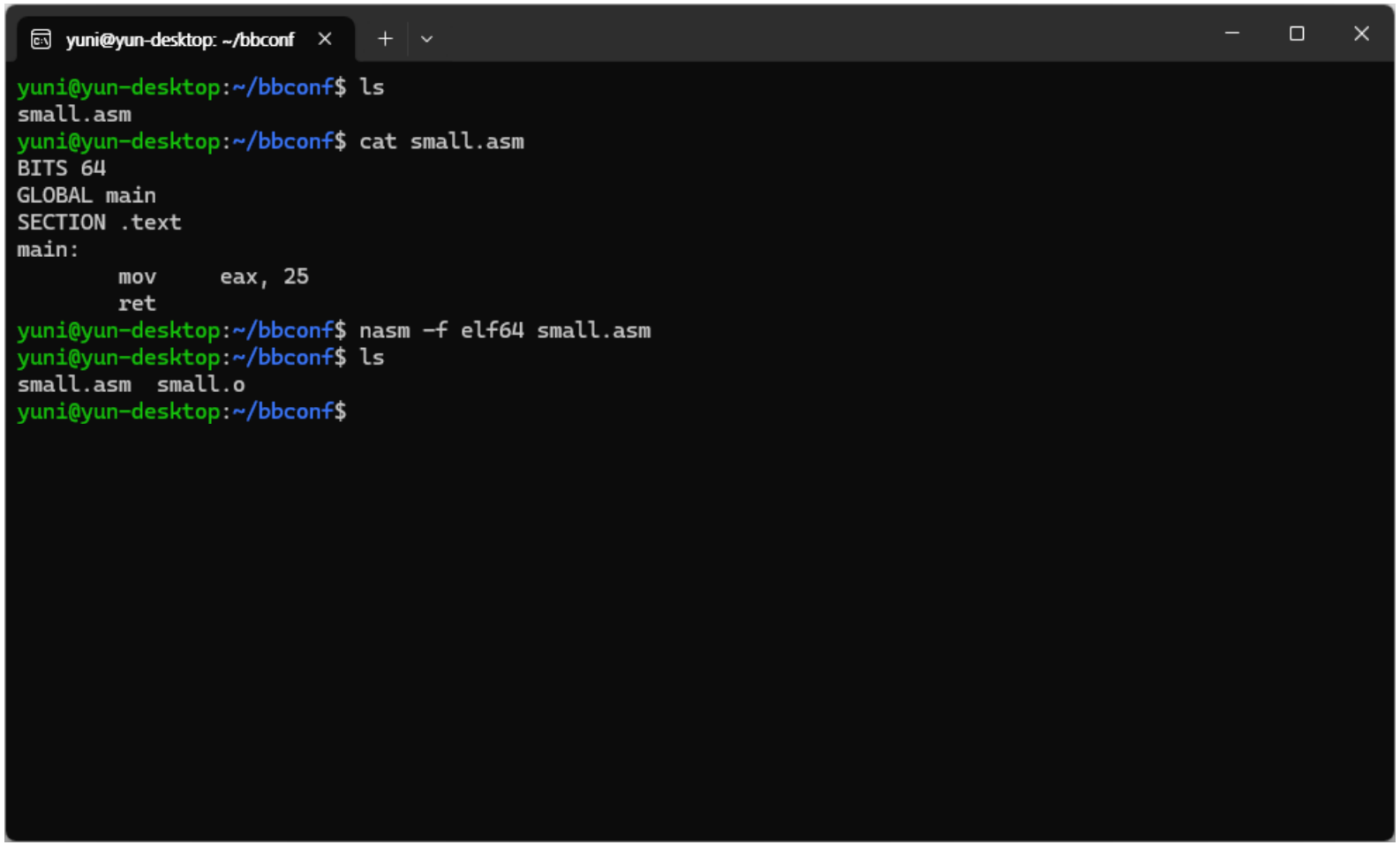
```
BITS 64
GLOBAL main
SECTION .text
main:
    mov     eax, 25
    ret
```



Best record **14328** Bytes

nasm으로 도전

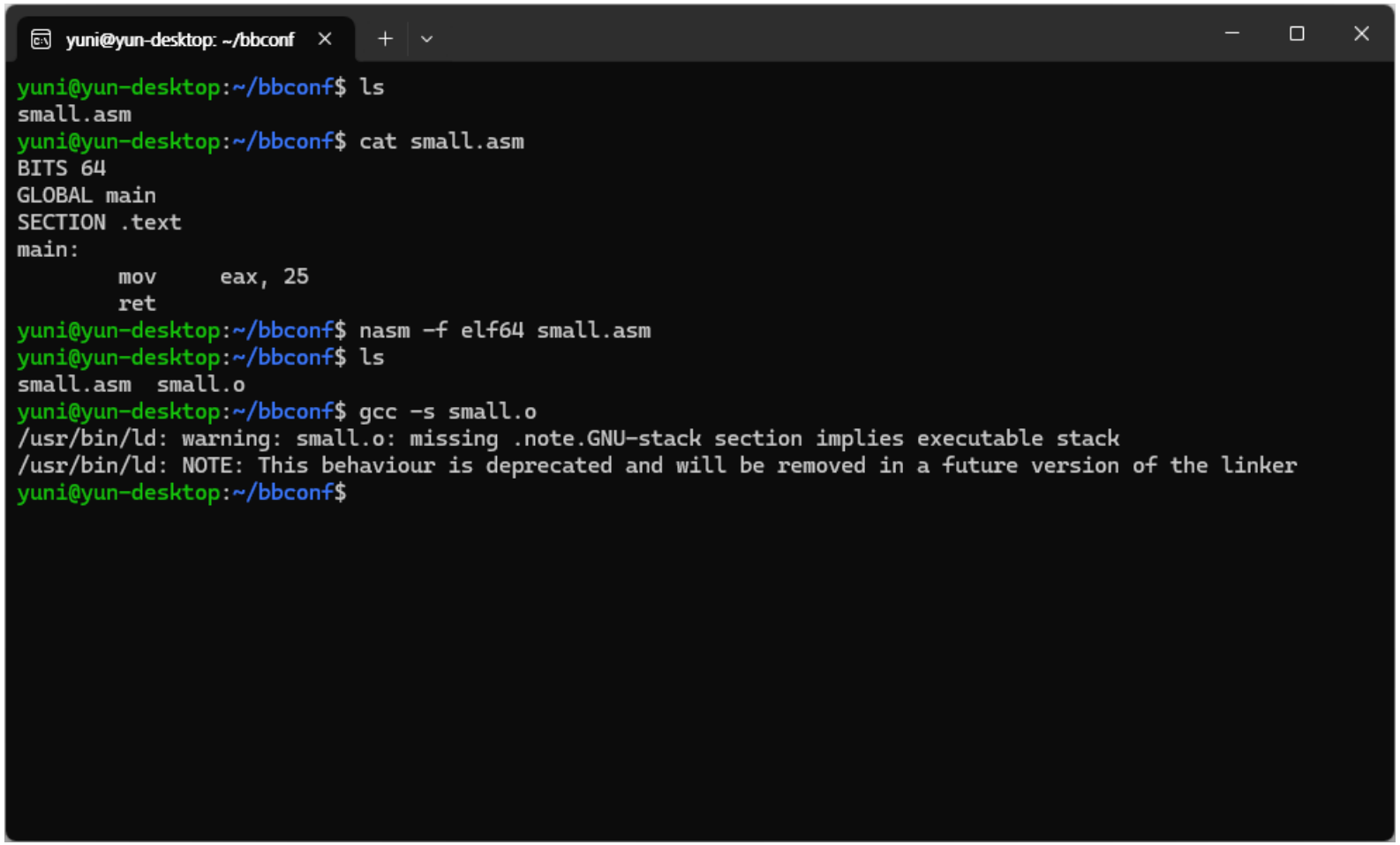
```
BITS 64
GLOBAL main
SECTION .text
main:
    mov     eax, 25
    ret
```



Best record **14328** Bytes

nasm으로 도전

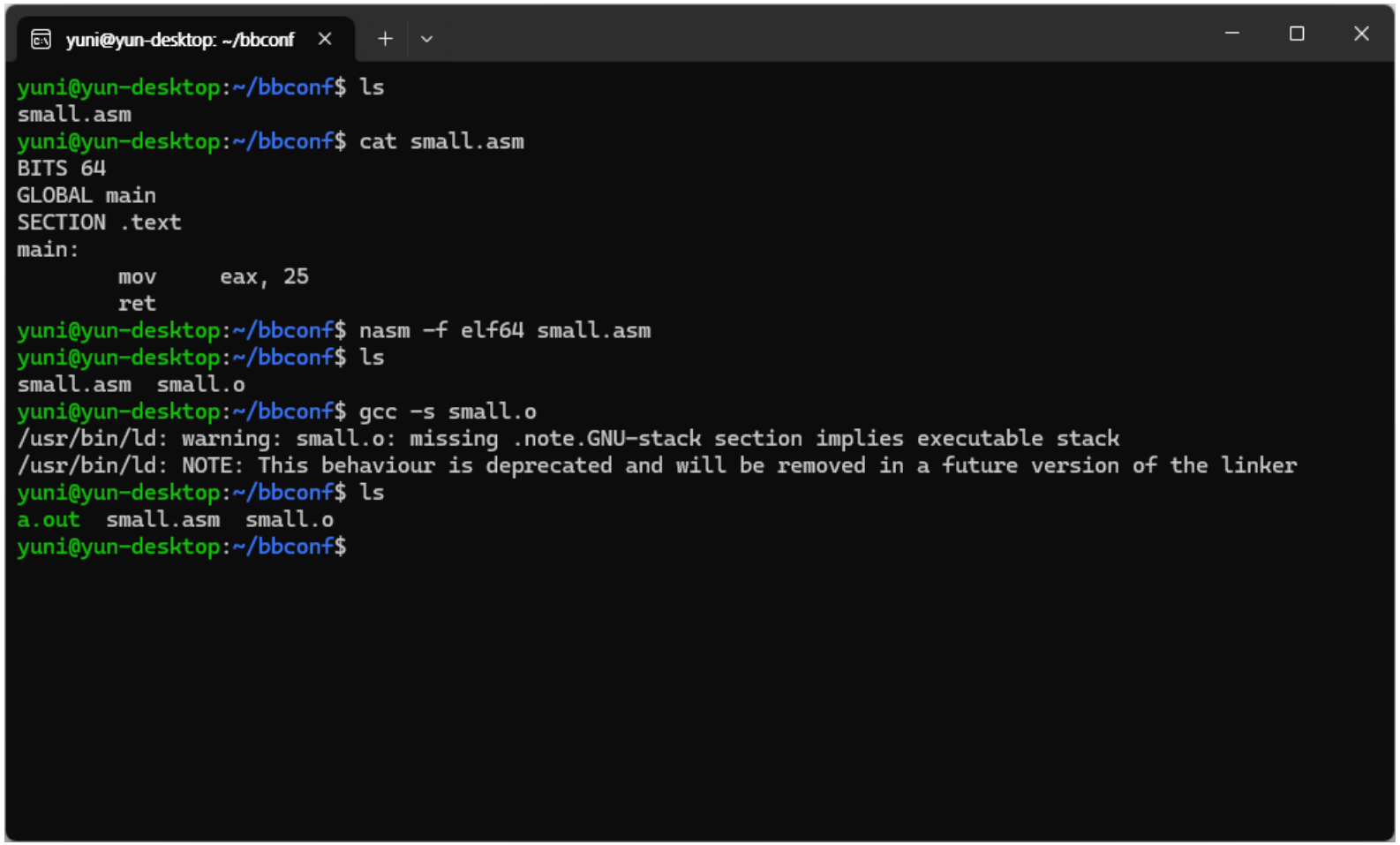
```
BITS 64
GLOBAL main
SECTION .text
main:
    mov     eax, 25
    ret
```



Best record **14328** Bytes

nasm으로 도전

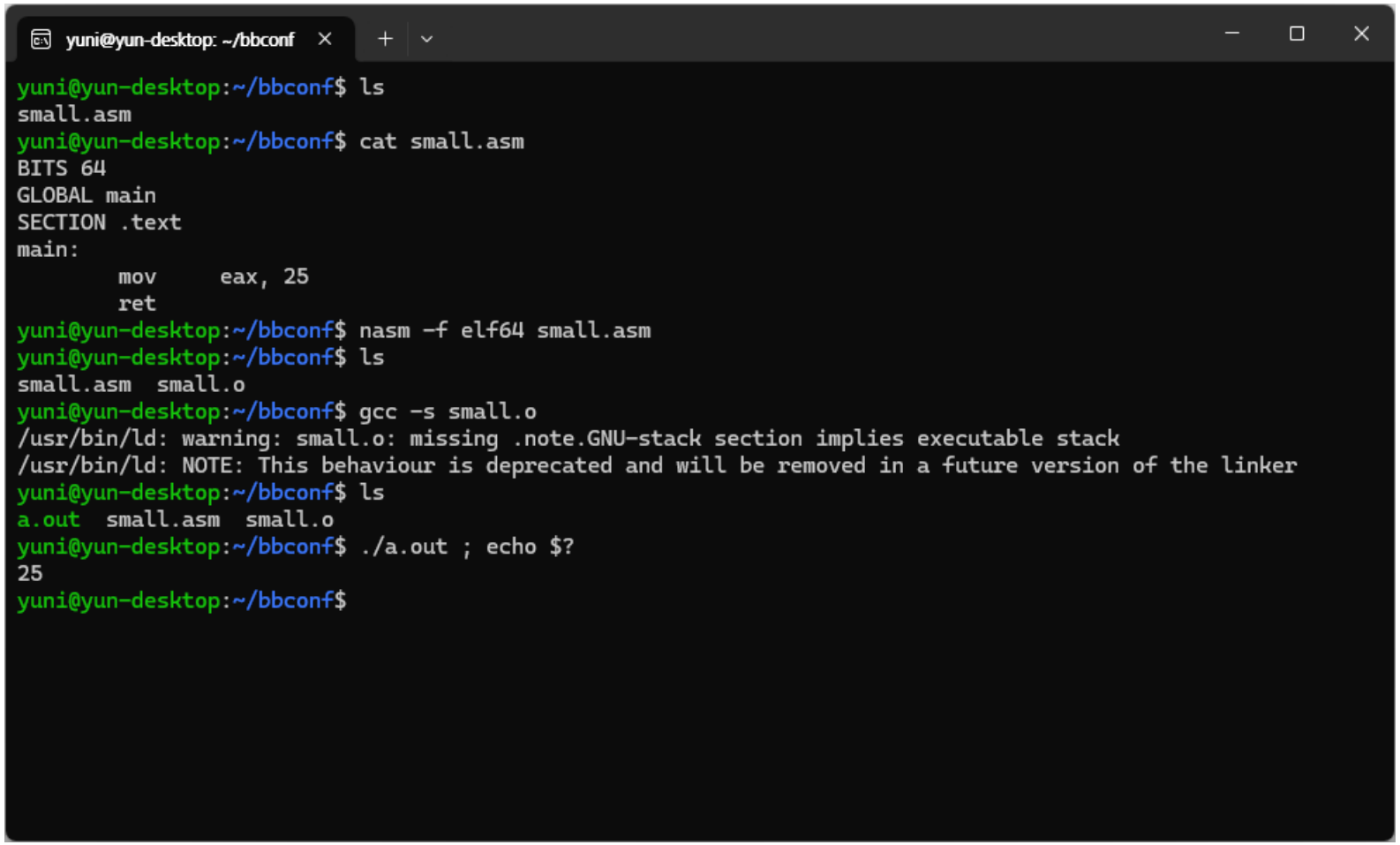
```
BITS 64
GLOBAL main
SECTION .text
main:
    mov     eax, 25
    ret
```



Best record **14328** Bytes

nasm으로 도전

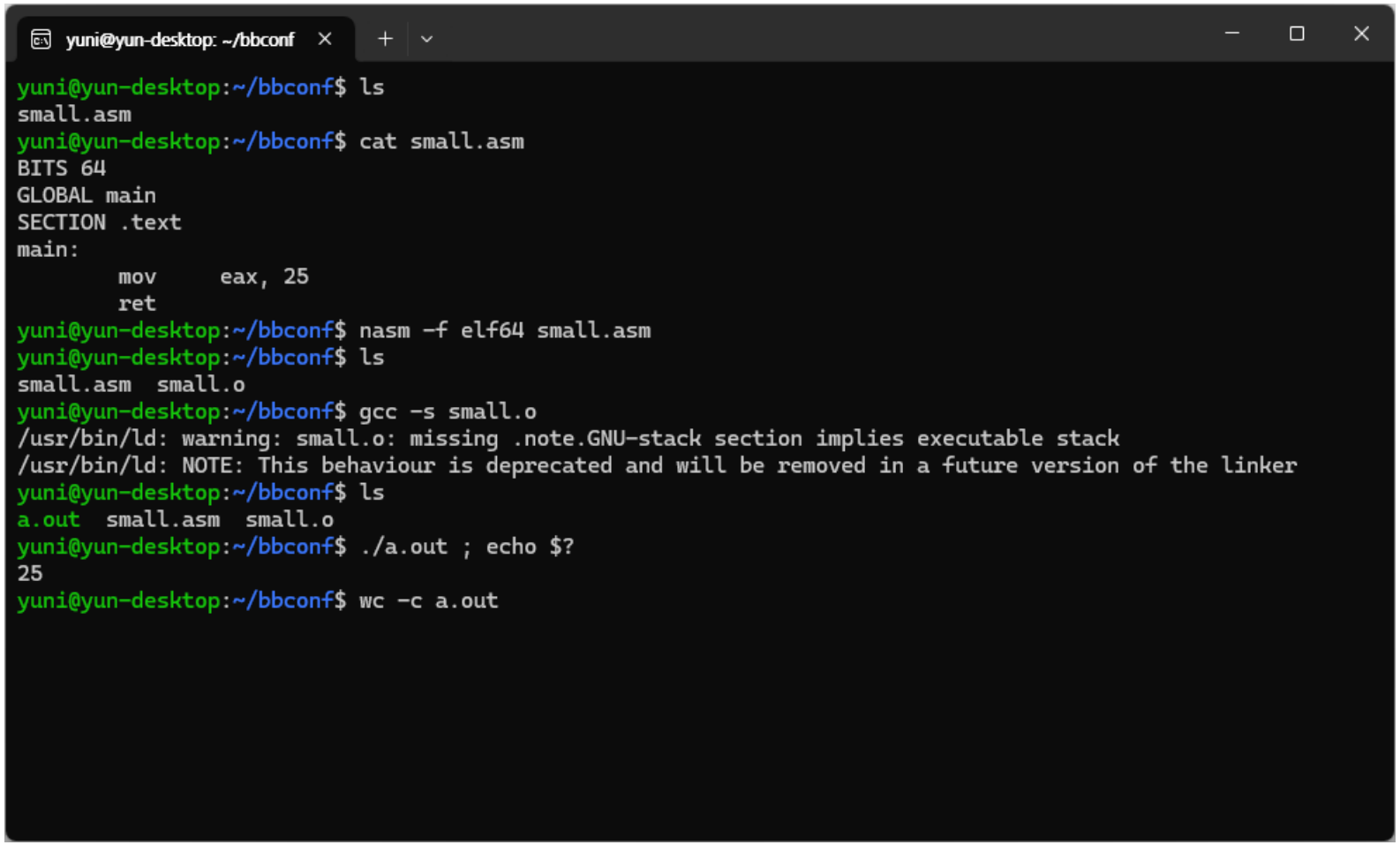
```
BITS 64
GLOBAL main
SECTION .text
main:
    mov     eax, 25
    ret
```



Best record **14328** Bytes

nasm으로 도전

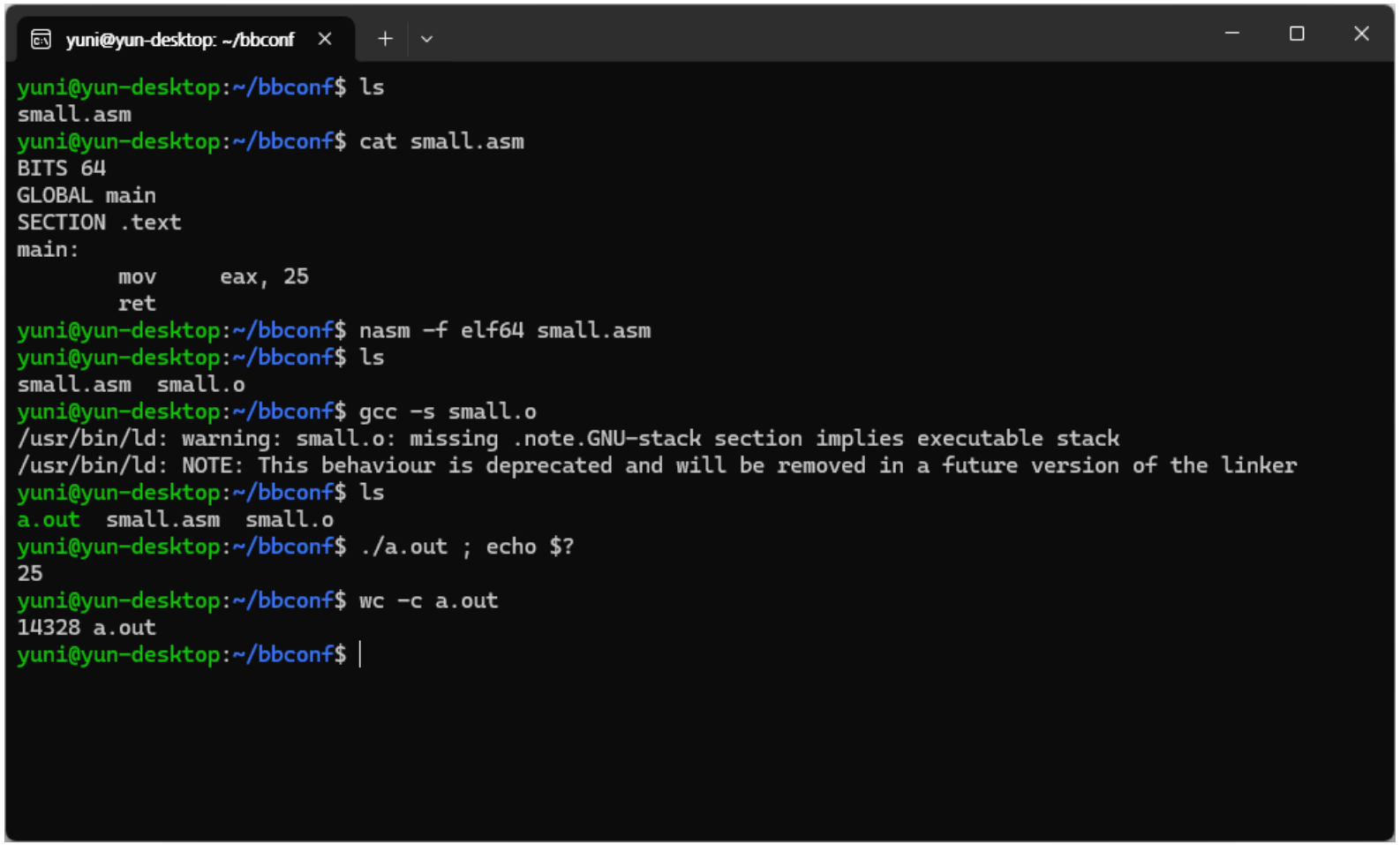
```
BITS 64
GLOBAL main
SECTION .text
main:
    mov     eax, 25
    ret
```



Best record **14328** Bytes

nasm으로 도전

```
BITS 64
GLOBAL main
SECTION .text
main:
    mov     eax, 25
    ret
```



Best record **14328** Bytes

main 함수를 꼭 써야하는가?

```
1 main;__libc_start_main(){_Exit(!write(1,"Hello World!",12));}
```

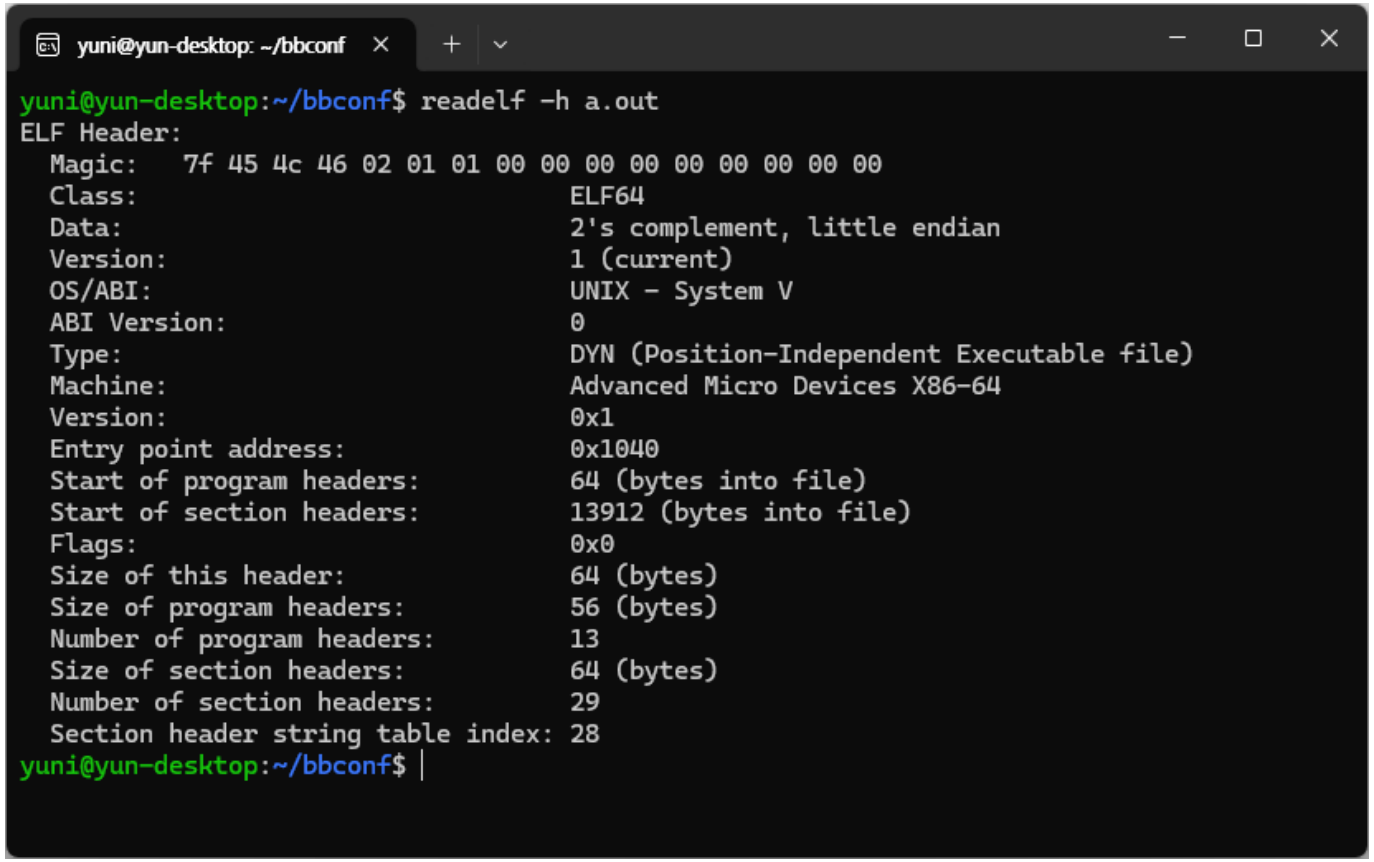
소스 코드 공개 ○ 공개 ○ 비공개 ◉ 맞았을 때만 공개 수정

복사 다운로드 공유

제출 번호	아이디	문제	문제 제목	결과	메모리	시간	언어	코드 길이	제출한 시간
35201497	YunGoon	2557	Hello World	맞았습니다!!	156 KB	0 ms	C99	61 B	3년 전

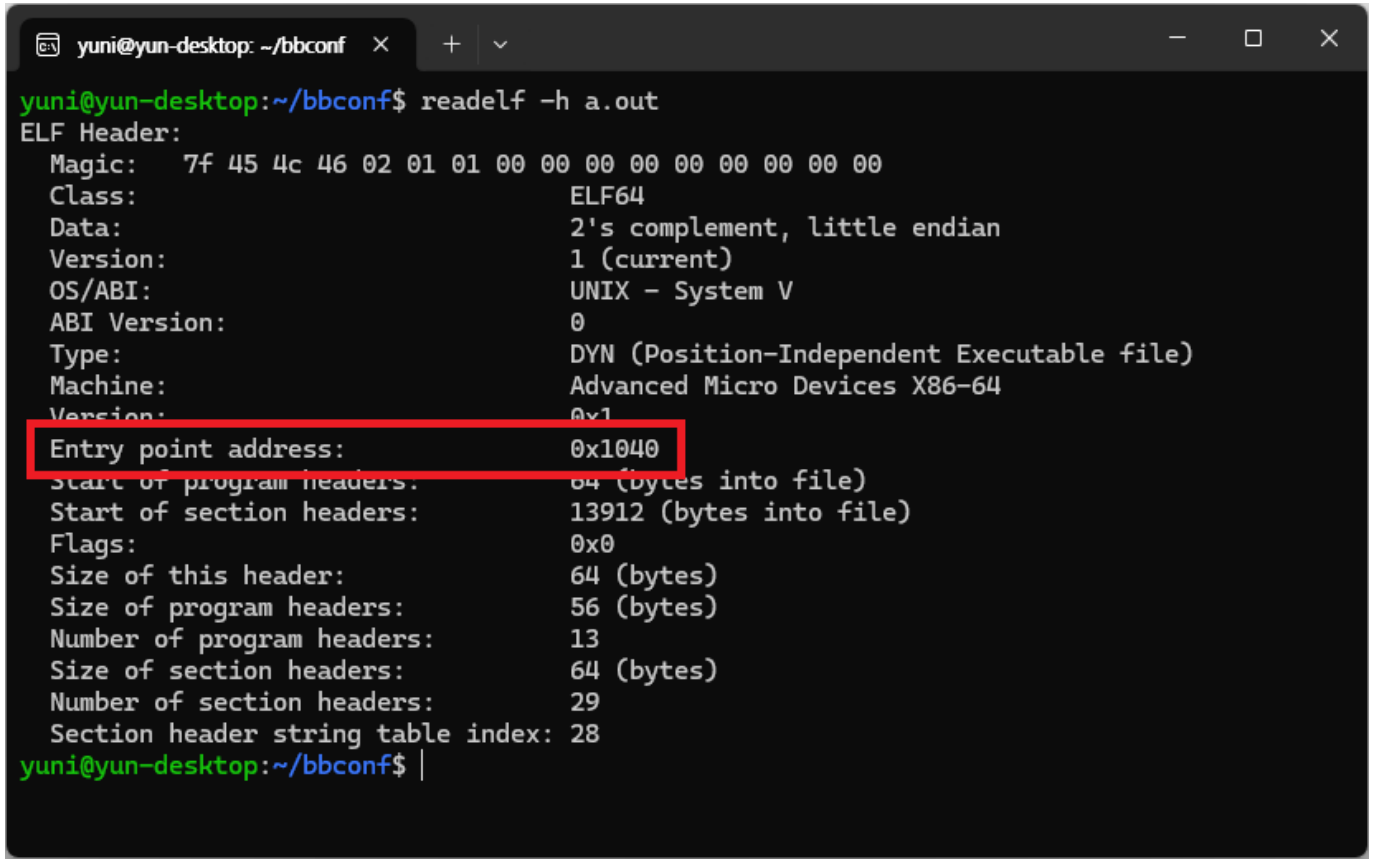
Best record **14328** Bytes

main 함수를 꼭 써야하는가?



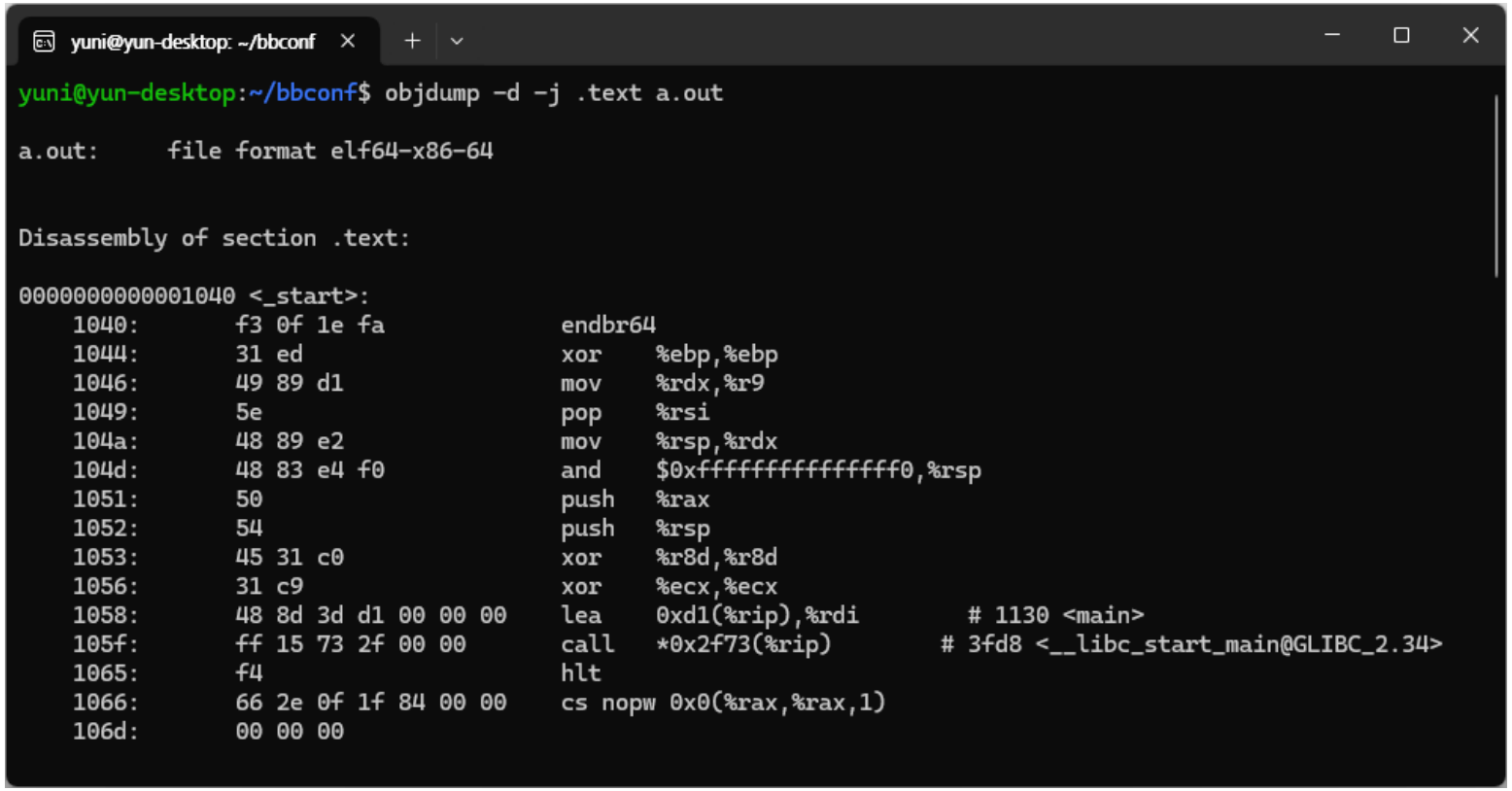
Best record **14328** Bytes

main 함수를 꼭 써야하는가?



Best record **14328** Bytes

main 함수를 꼭 써야하는가?



Best record **14328** Bytes

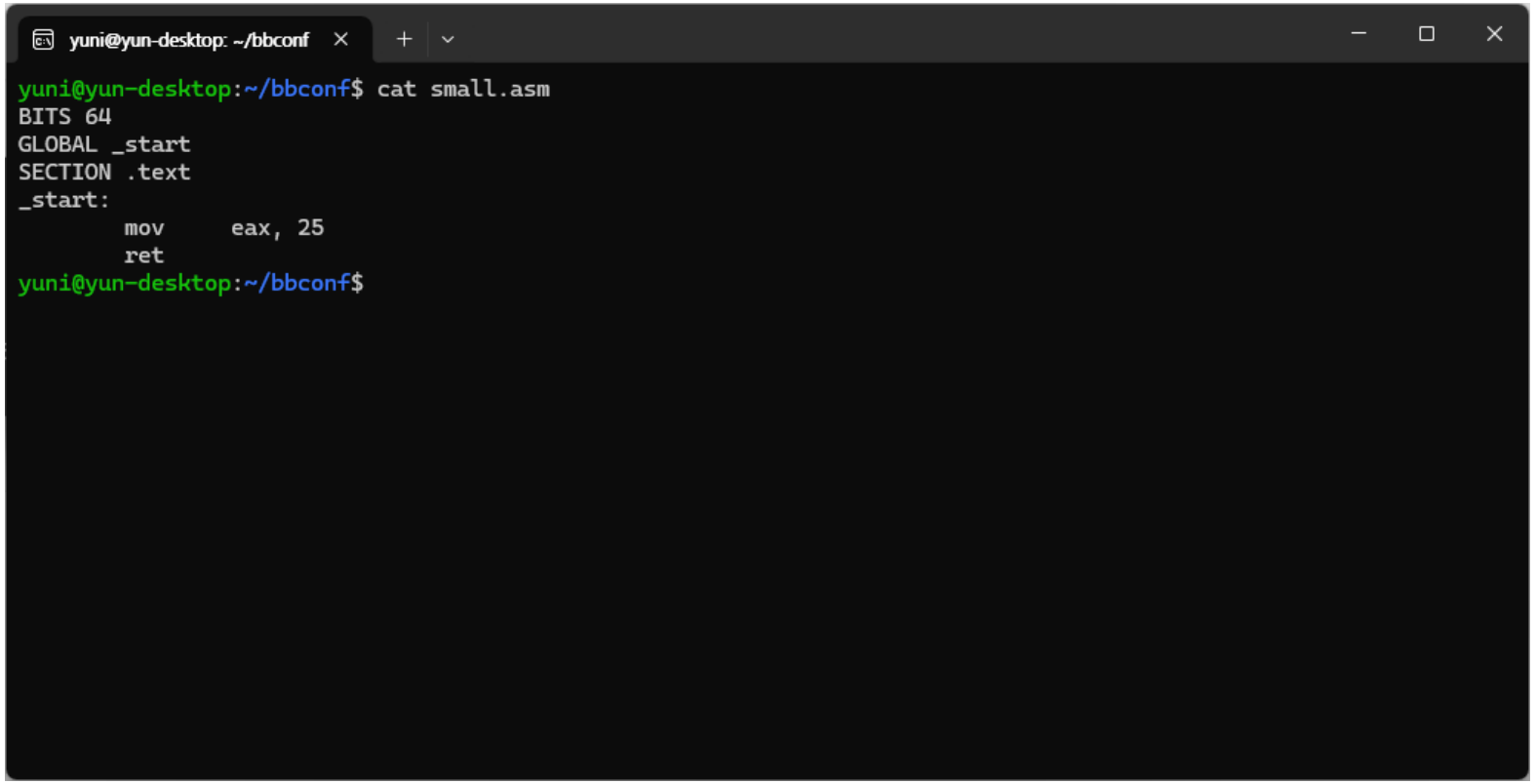
main 함수를 꼭 써야하는가?

```
BITS 64
GLOBAL _start
SECTION .text
_start:
    mov     eax, 25
    ret
```

Best record **14328** Bytes

main 함수를 꼭 써야하는가?

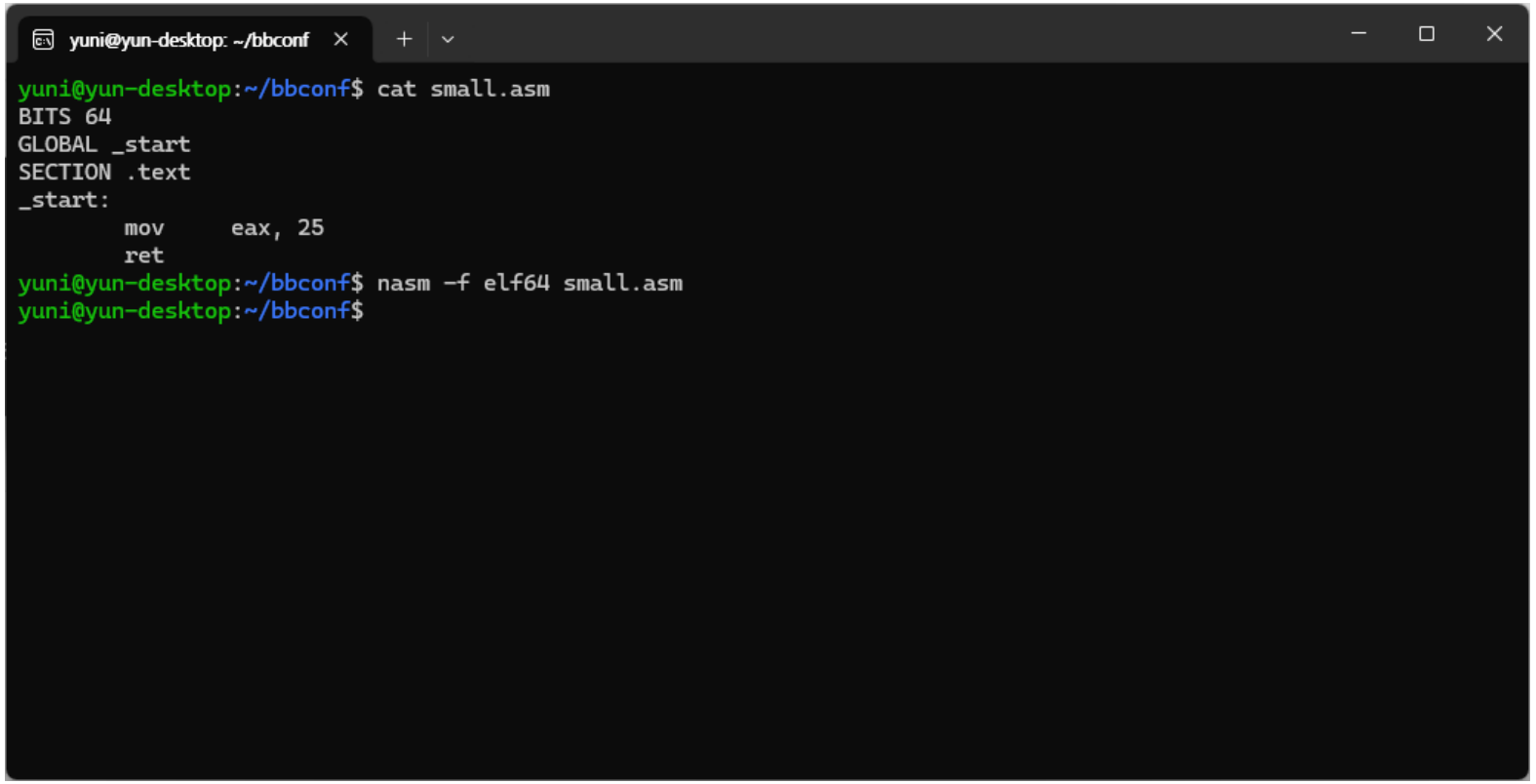
```
BITS 64
GLOBAL _start
SECTION .text
_start:
    mov     eax, 25
    ret
```



Best record **14328** Bytes

main 함수를 꼭 써야하는가?

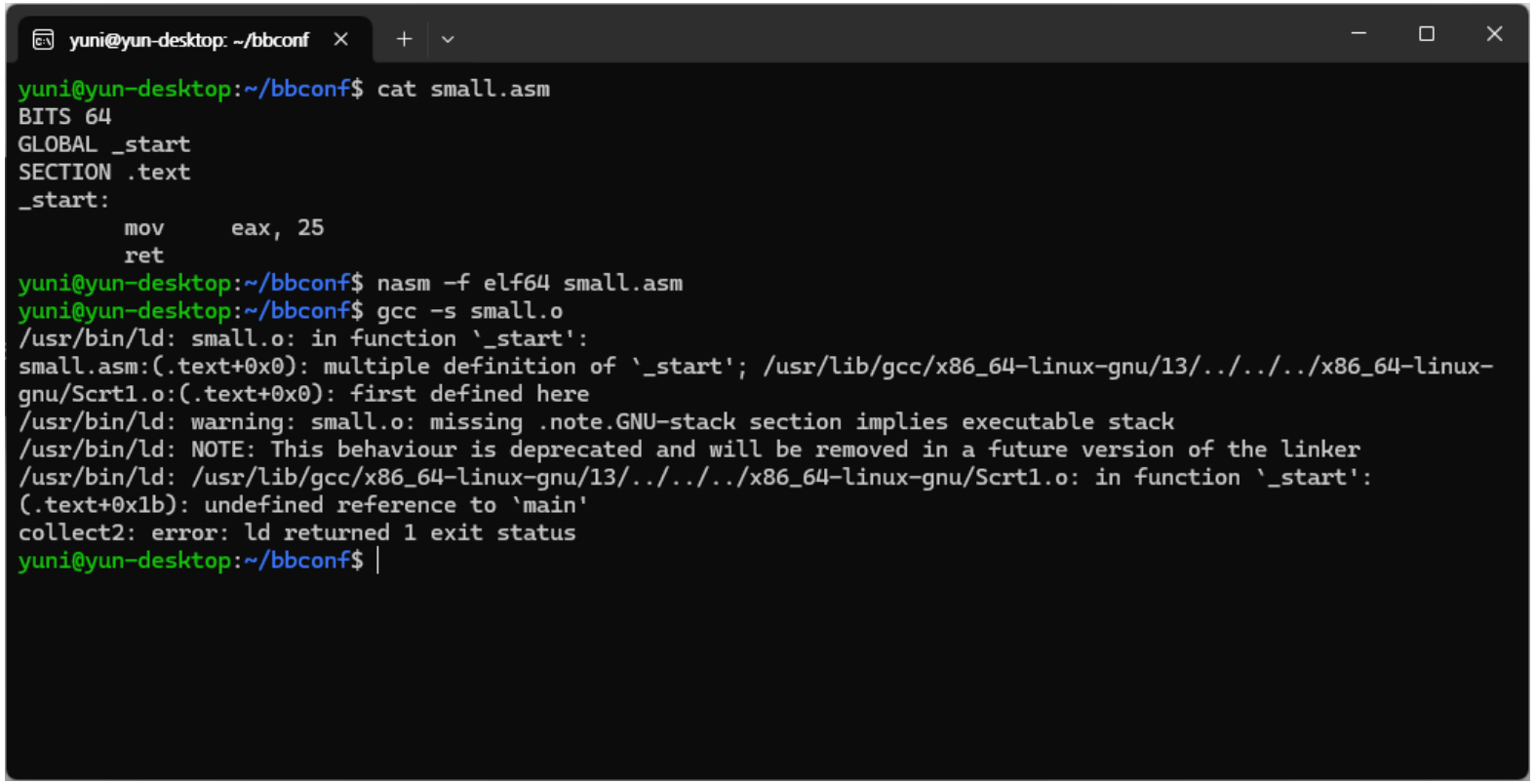
```
BITS 64
GLOBAL _start
SECTION .text
_start:
    mov     eax, 25
    ret
```



Best record **14328** Bytes

main 함수를 꼭 써야하는가?

```
BITS 64
GLOBAL _start
SECTION .text
_start:
    mov     eax, 25
    ret
```



Best record **14328** Bytes

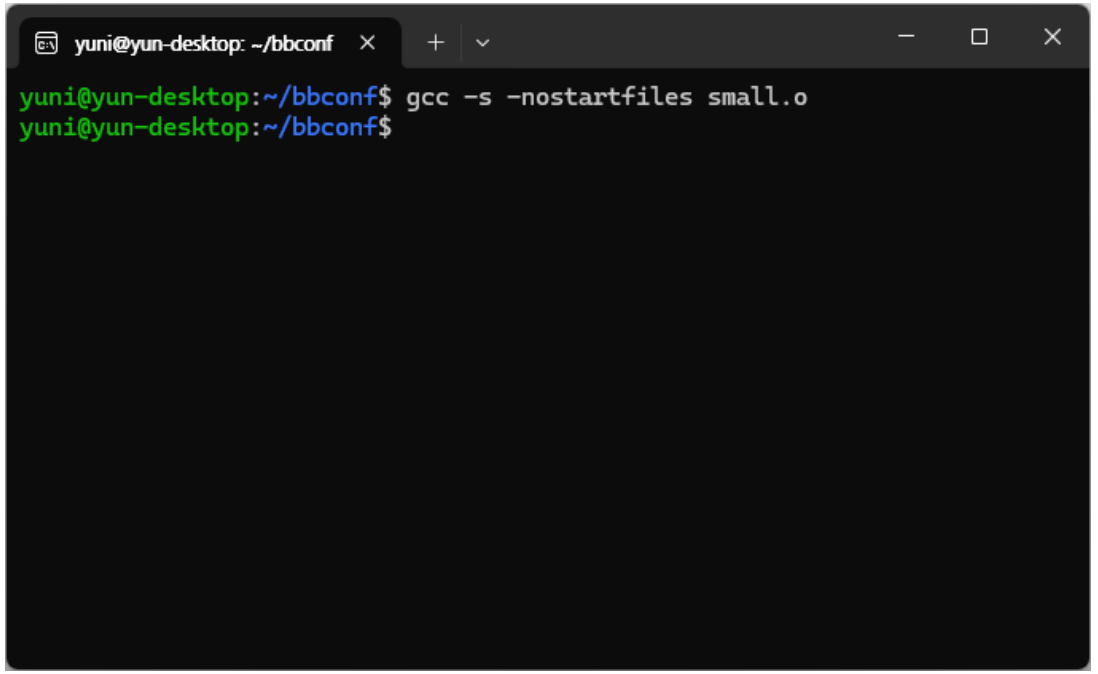
main 함수를 꼭 써야하는가?

```
-nostartfiles
    Do not use the standard system startup files when linking.
    The standard system libraries are used normally, unless
    -nostdlib, -nolibc, or -nodefaultlibs is used.
```

Best record **14328** Bytes

main 함수를 꼭 써야하는가?

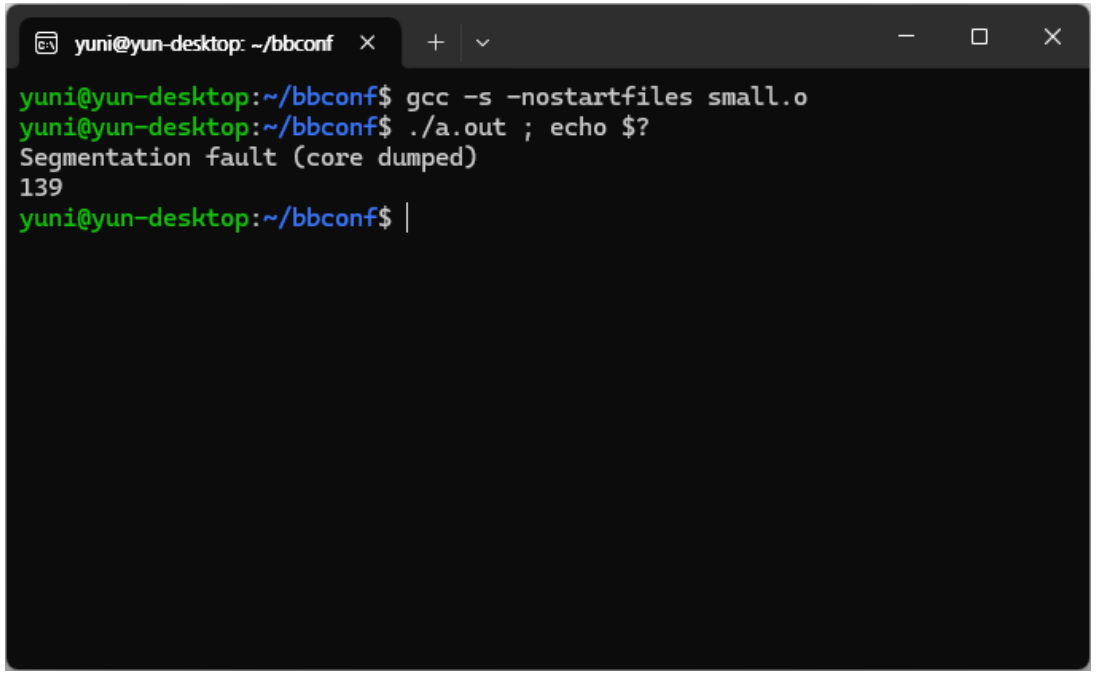
```
BITS 64
GLOBAL _start
SECTION .text
_start:
    mov     eax, 25
    ret
```



Best record **14328** Bytes

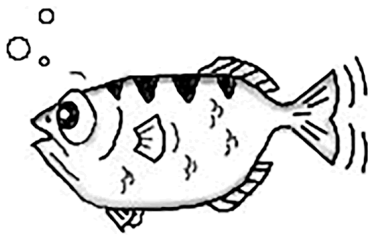
main 함수를 꼭 써야하는가?

```
BITS 64
GLOBAL _start
SECTION .text
_start:
    mov     eax, 25
    ret
```



Best record **14328** Bytes

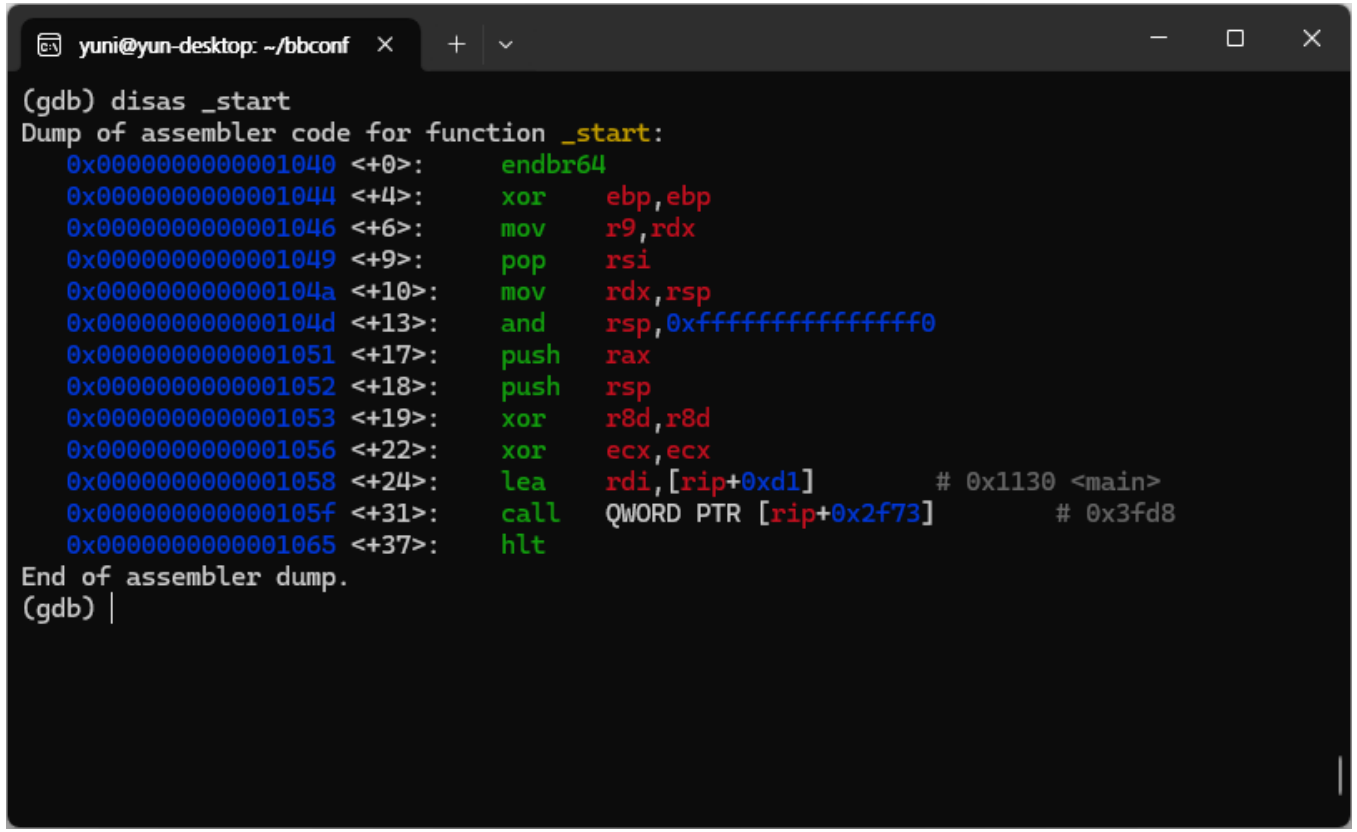
_start의 정체



GDB
The GNU Project
Debugger

Best record **14328** Bytes

_start의 정체



Best record **14328** Bytes

_start의 정체

yuni@yun-desktop: ~/bbconf

```
(gdb) disas _start
Dump of assembler code for function _start:
   0x0000000000001040 <+0>:     endbr64
   0x0000000000001044 <+4>:     xor    ebp,ebp
   0x0000000000001046 <+6>:     mov    r9,rdx
   0x0000000000001049 <+9>:     pop    rsi
   0x000000000000104a <+10>:    mov    rdx,rsp
   0x000000000000104d <+13>:    and    rsp,0xfffffffffffffff0
   0x0000000000001051 <+17>:    push   rax
   0x0000000000001052 <+18>:    push   rsp
   0x0000000000001053 <+19>:    xor    r8d,r8d
   0x0000000000001056 <+22>:    xor    ecx,ecx
   0x0000000000001058 <+24>:    lea    rdi,[rip+0xd1]        # 0x1130 <main>
   0x000000000000105f <+31>:    call   QWORD PTR [rip+0x2f73]        # 0x3fd8
   0x0000000000001065 <+37>:    hlt
End of assembler dump.
(gdb)
```

Best record **14328** Bytes

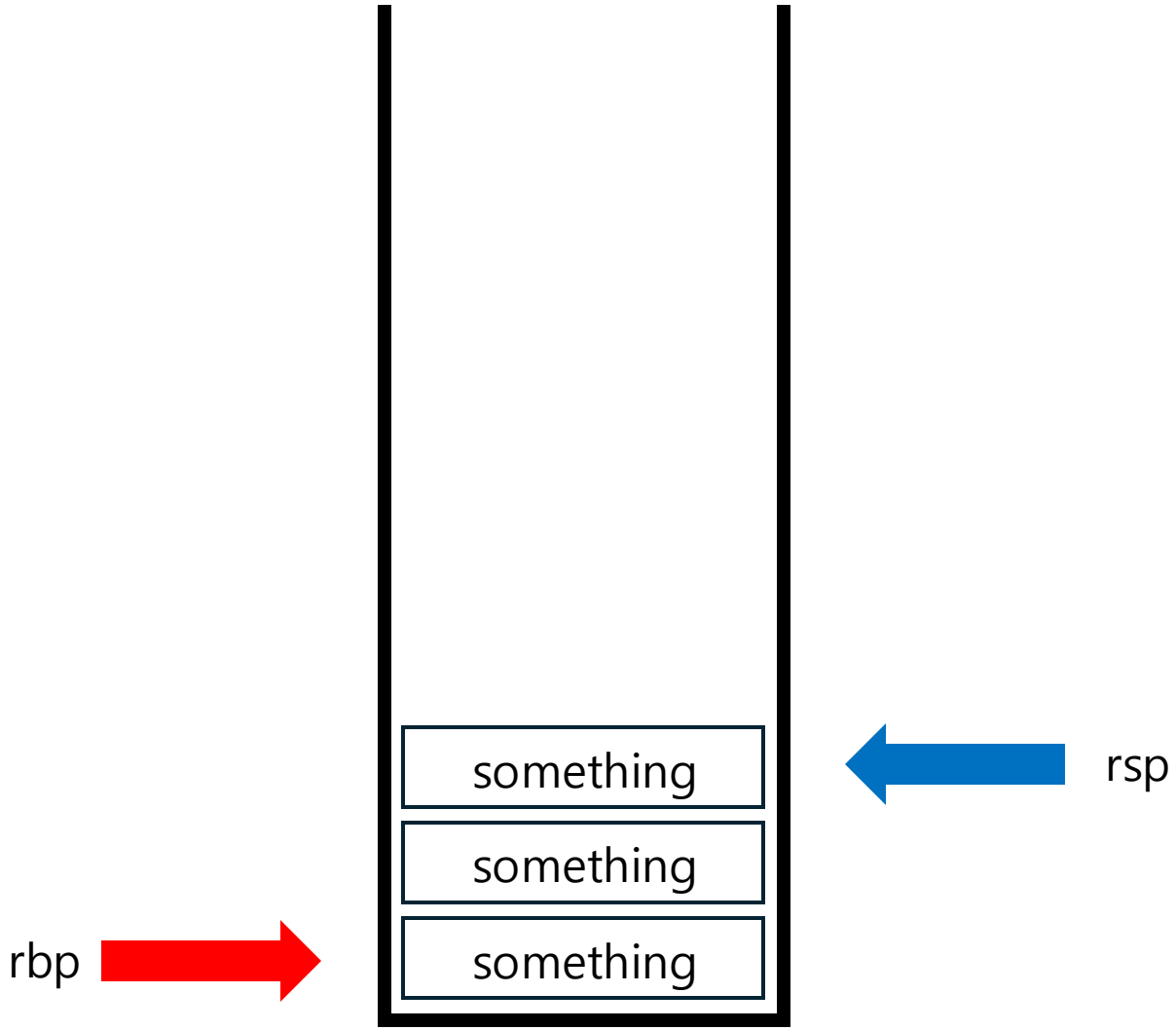
_start의 정체

이름	주용도
rax (accumulator register)	함수의 반환 값
rbx (base register)	x64에서는 주된 용도 없음
rcx (counter register)	반복문의 반복 횟수, 각종 연산의 시행 횟수
rdx (data register)	x64에서는 주된 용도 없음
rsi (source index)	데이터를 옮길 때 원본을 가리키는 포인터
rdi (destination index)	데이터를 옮길 때 목적지를 가리키는 포인터
rsp (stack pointer)	사용중인 스택의 위치를 가리키는 포인터
rbp (stack base pointer)	스택의 바닥을 가리키는 포인터

Best record **14328** Bytes

_start의 정체

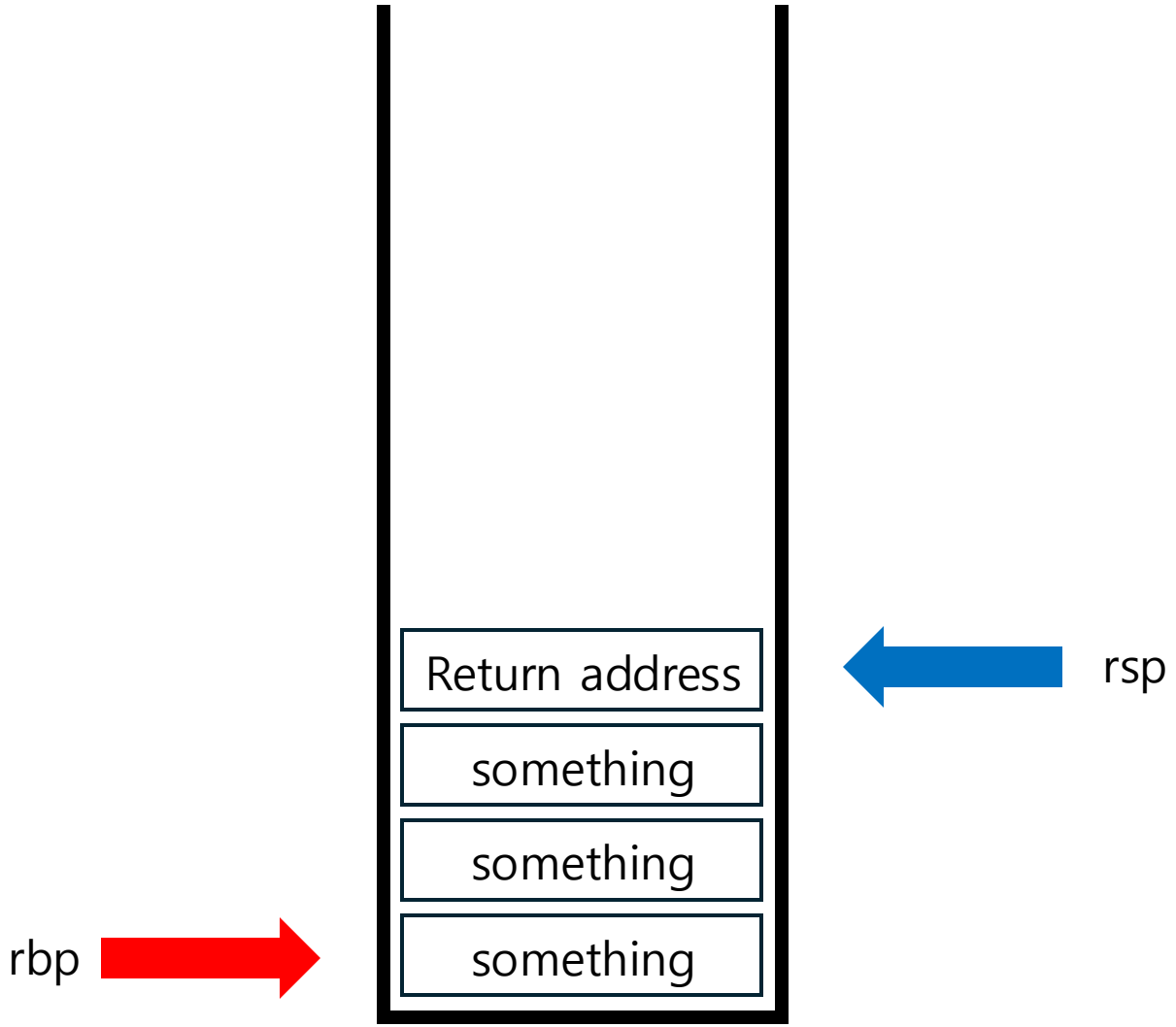
```c
int main() {
    int x = 1234;
    return 0;
}
```



Best record **14328** Bytes

_start의 정체

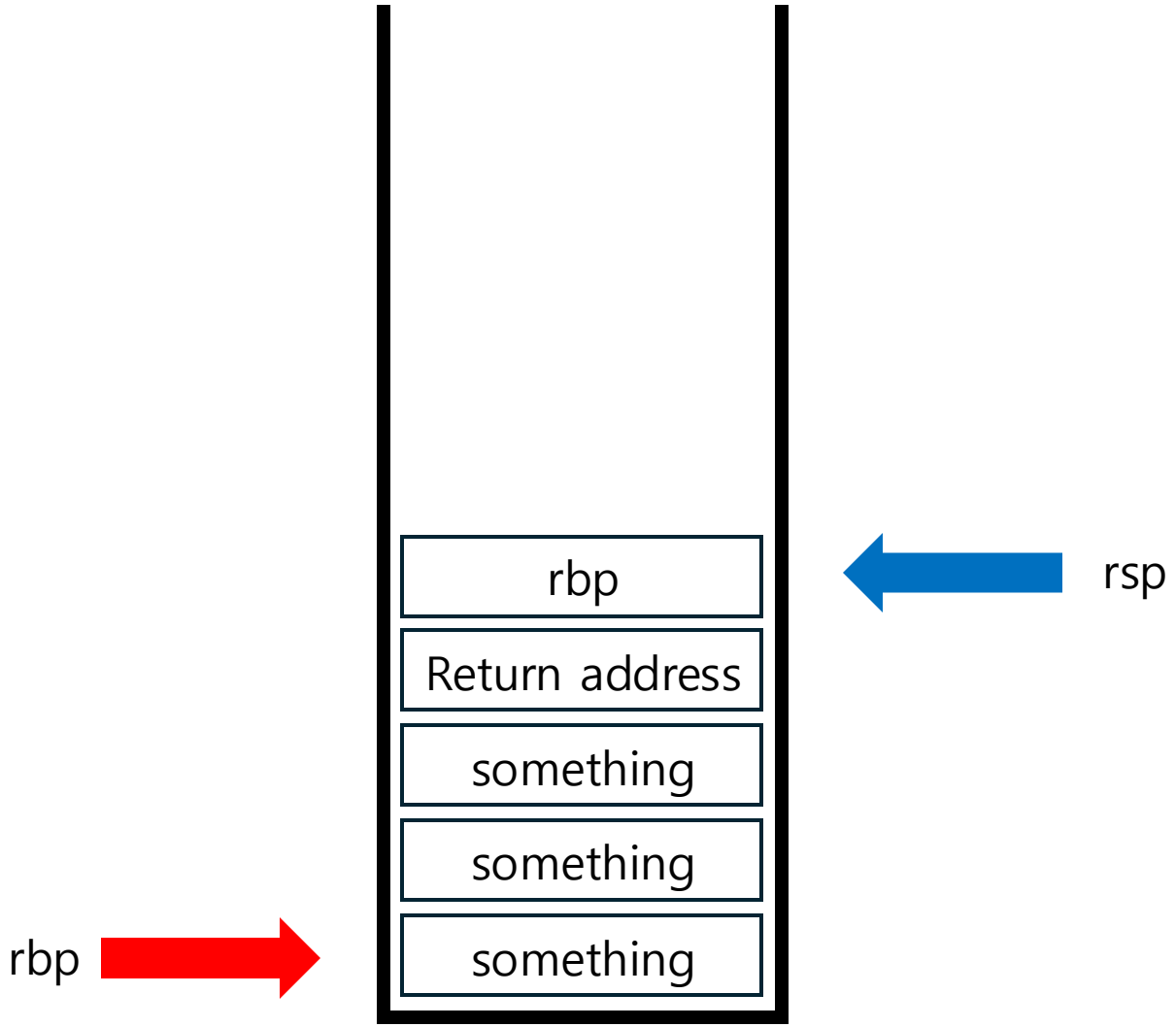
```c
int main() {
    int x = 1234;
    return 0;
}
```



Best record **14328** Bytes

_start의 정체

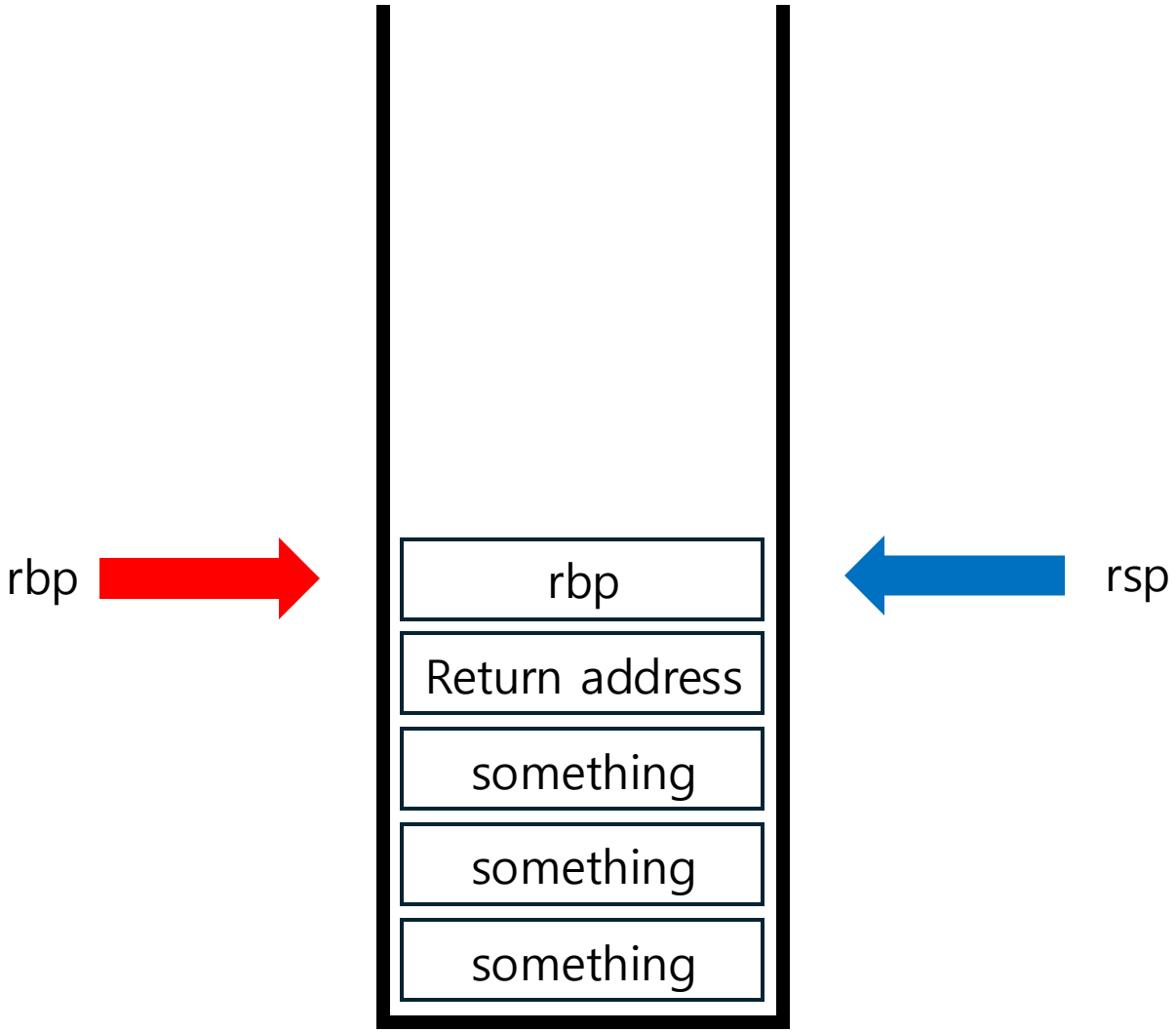
```
int main() {
    int x = 1234;
    return 0;
}
```



Best record **14328** Bytes

_start의 정체

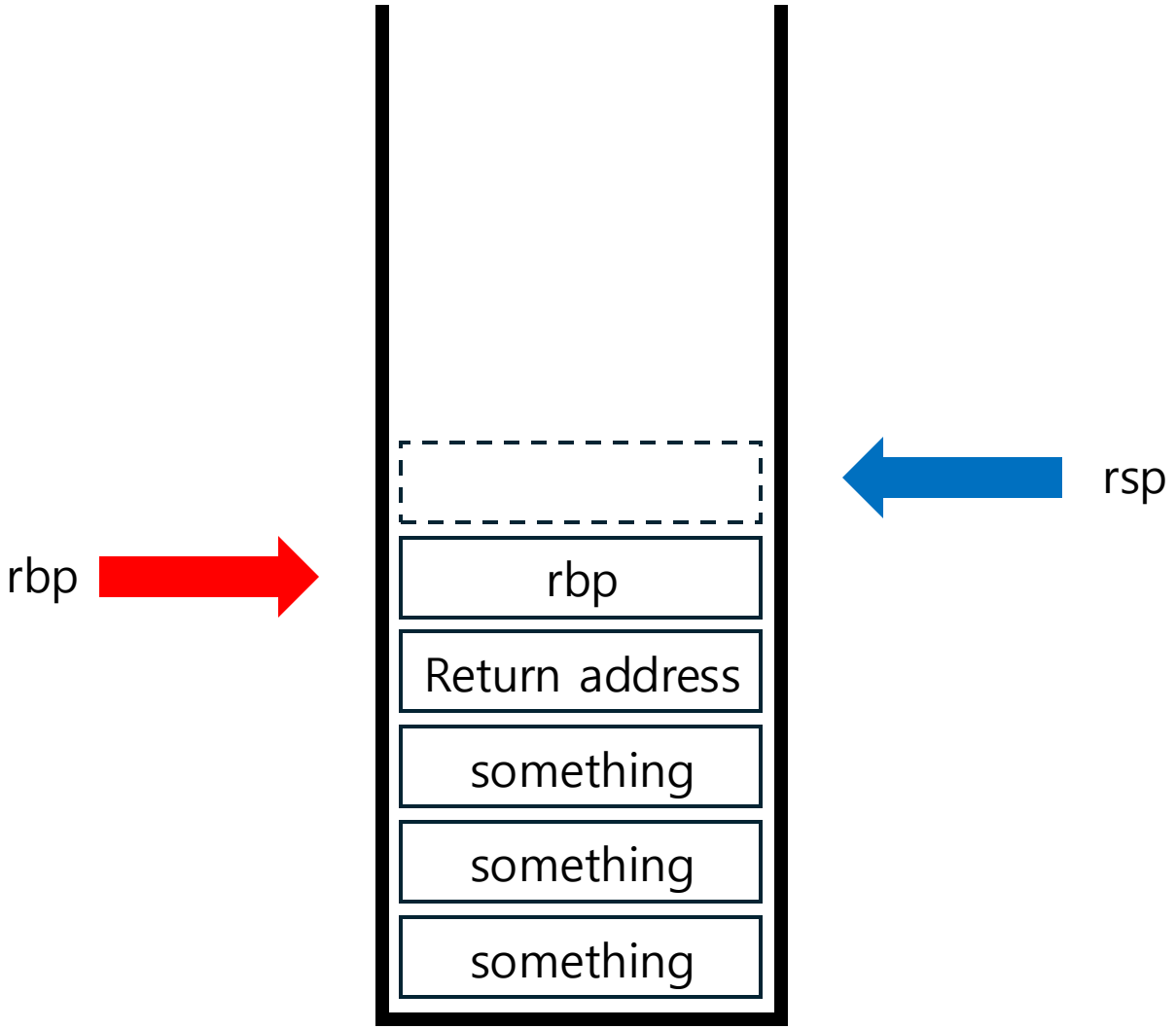
```c
int main() {
    int x = 1234;
    return 0;
}
```



Best record **14328** Bytes

_start의 정체

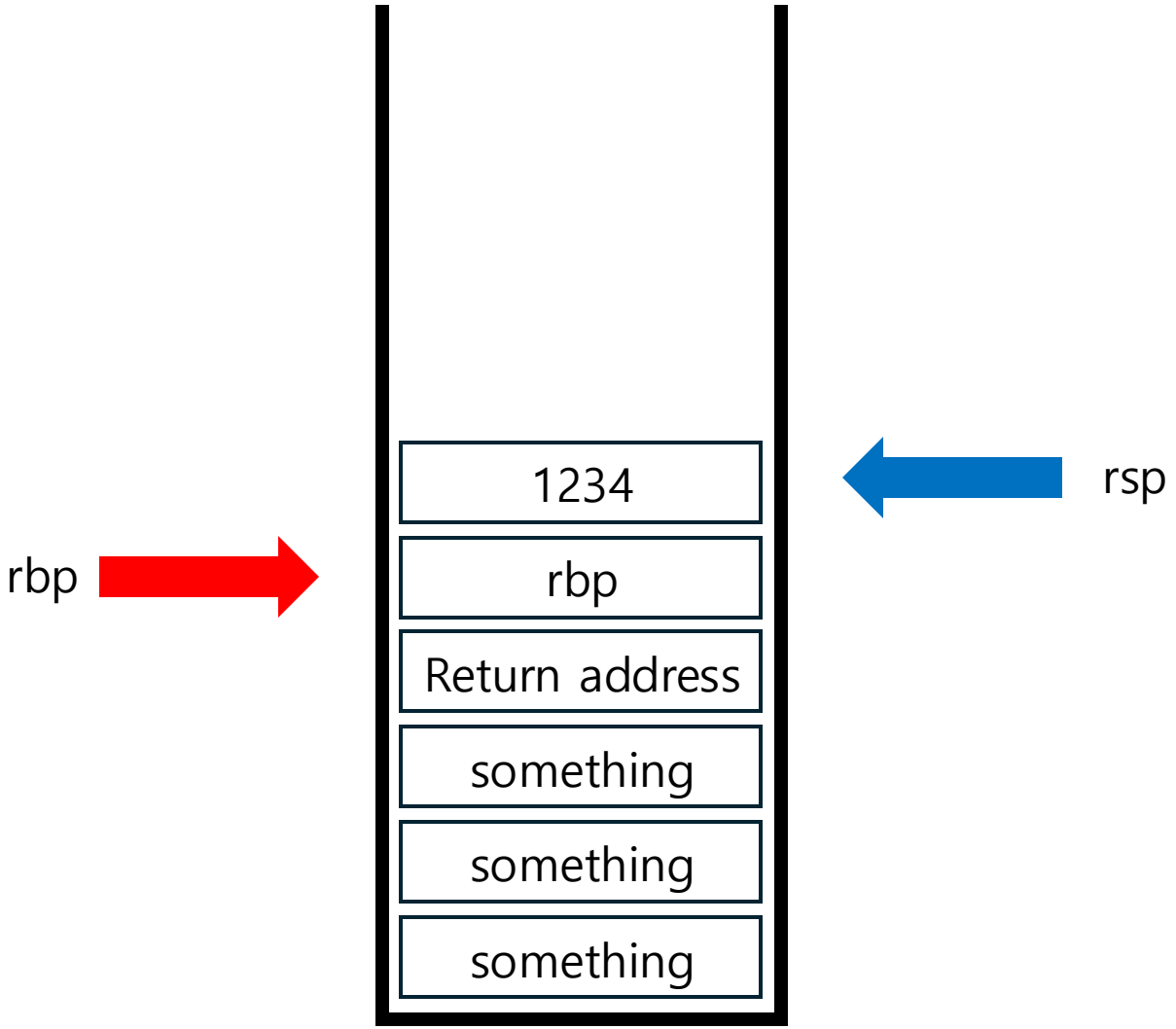
```c
int main() {
    int x = 1234;
    return 0;
}
```



Best record **14328** Bytes

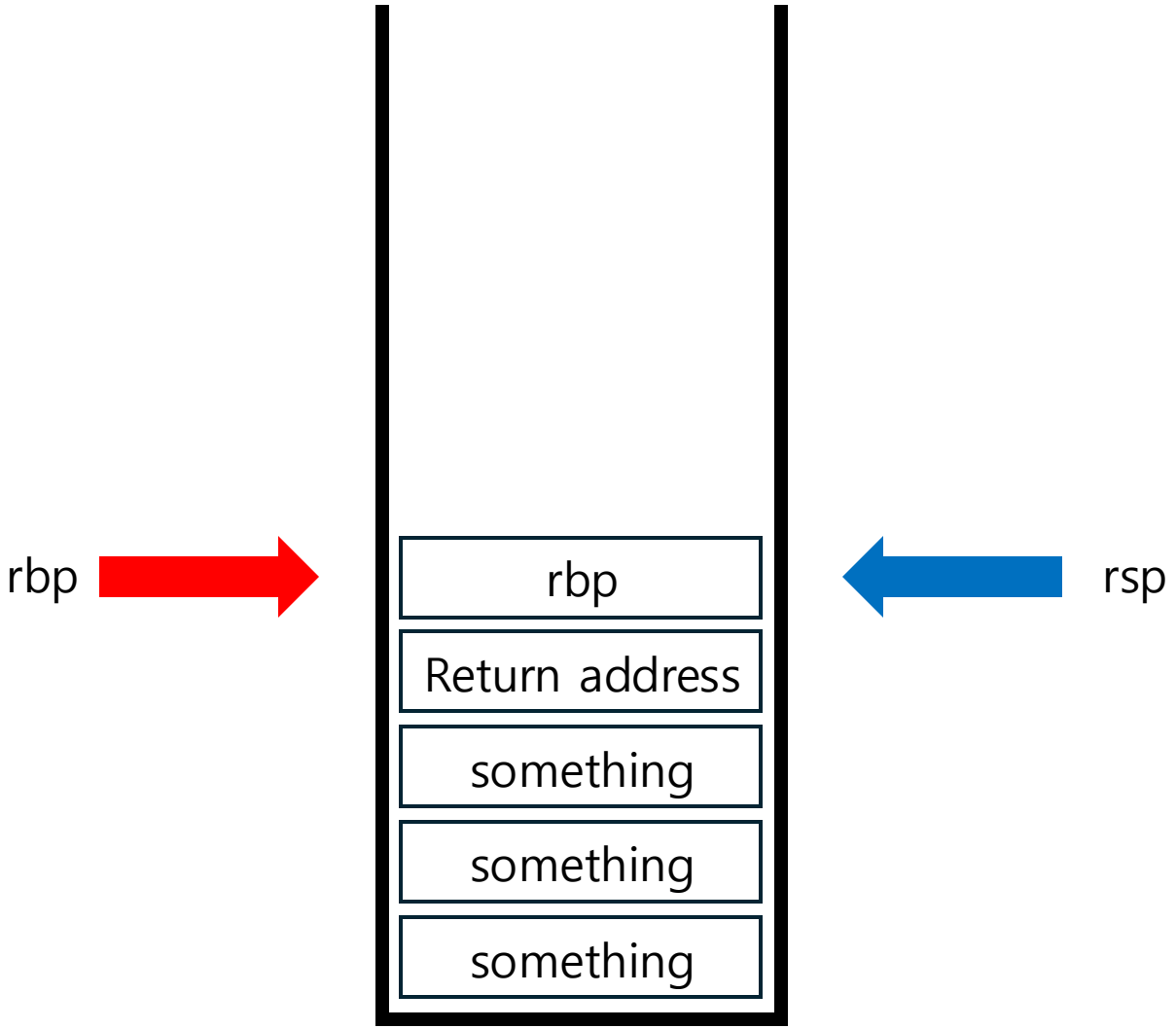
_start의 정체

```c
int main() {
    int x = 1234;
    return 0;
}
```



_start의 정체

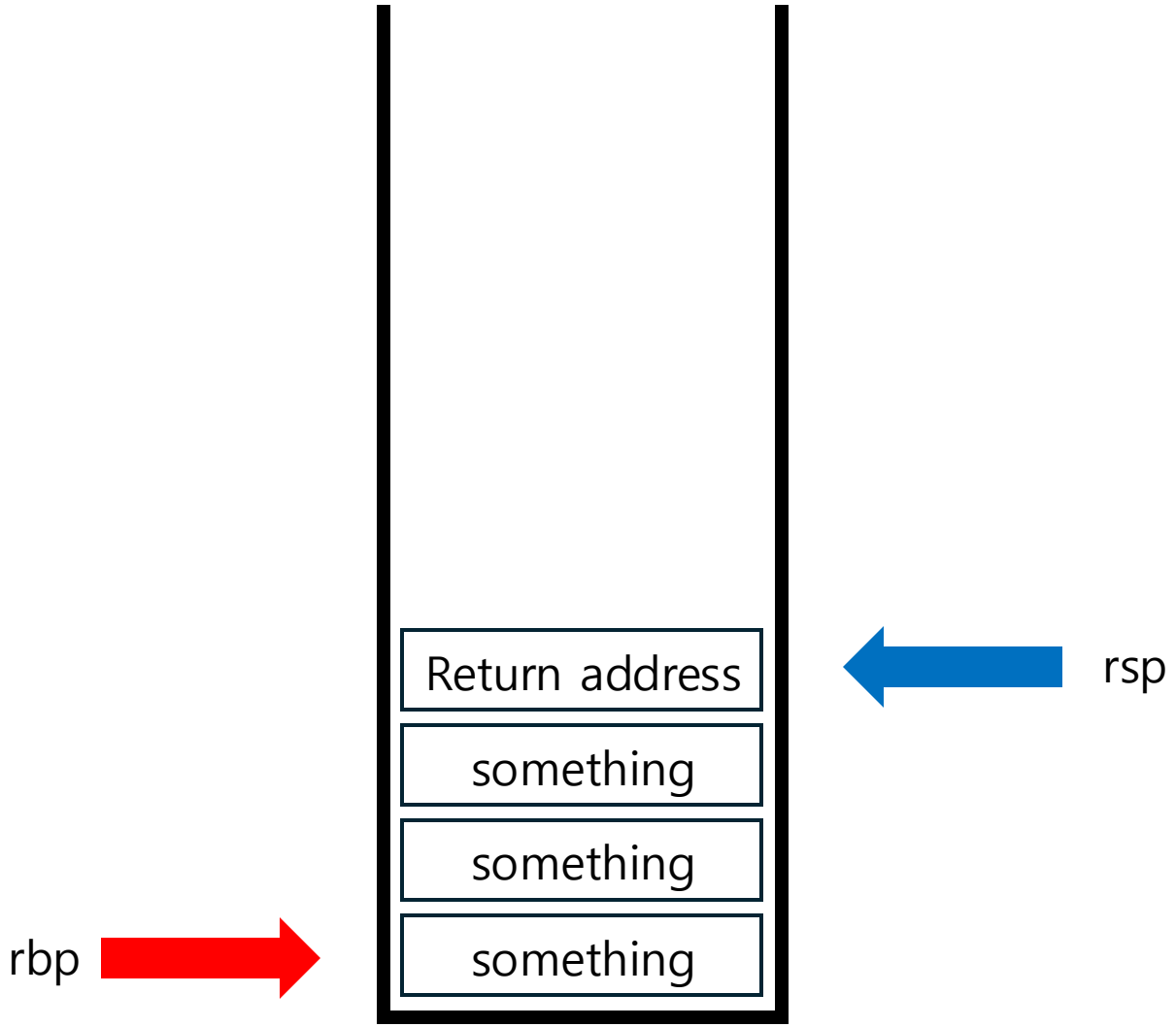
```c
int main() {
    int x = 1234;
    return 0;
}
```



Best record **14328** Bytes

_start의 정체

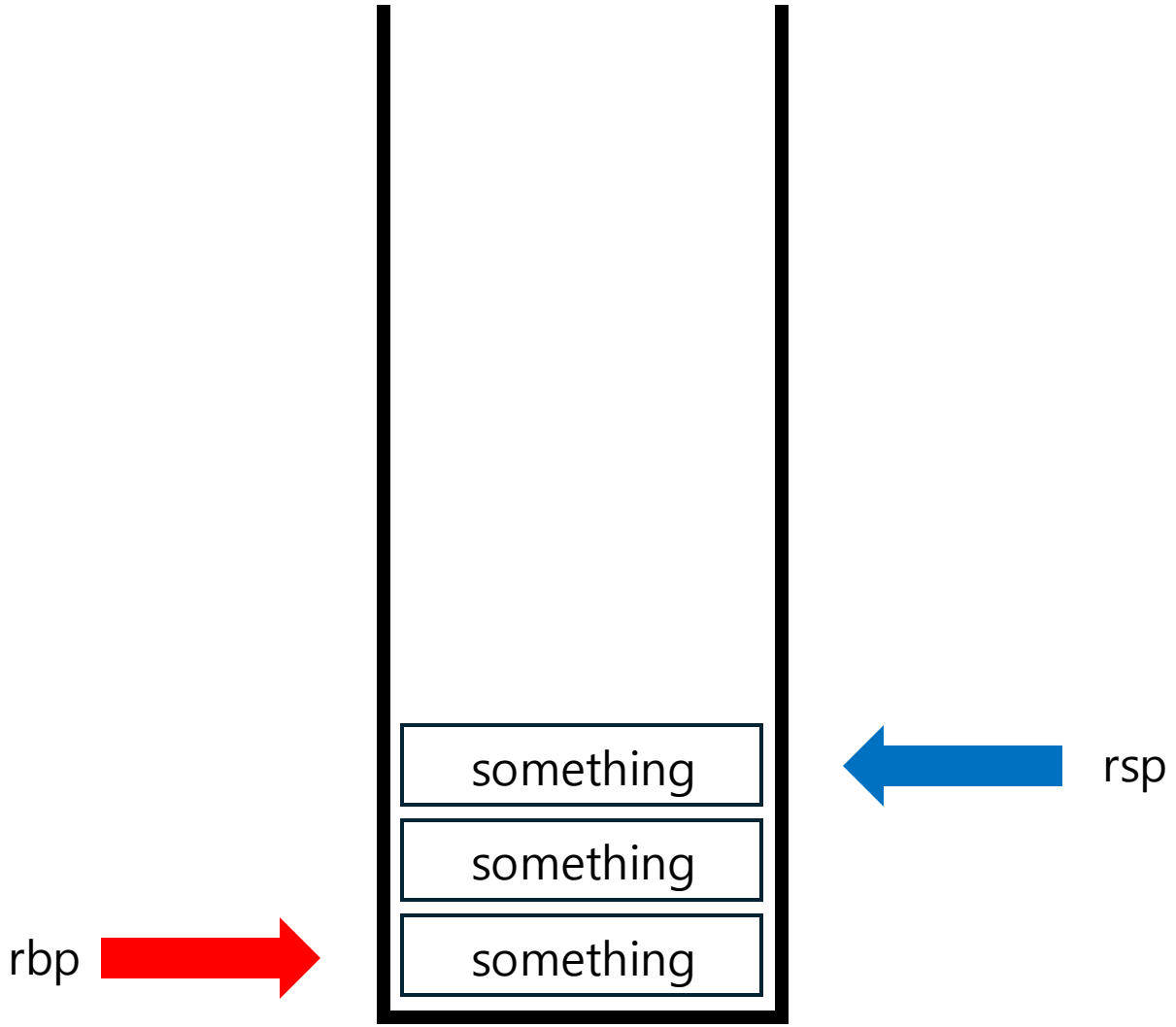
```c
int main() {
    int x = 1234;
    return 0;
}
```



Best record **14328** Bytes

_start의 정체

```c
int main() {
    int x = 1234;
    return 0;
}
```



Best record **14328** Bytes

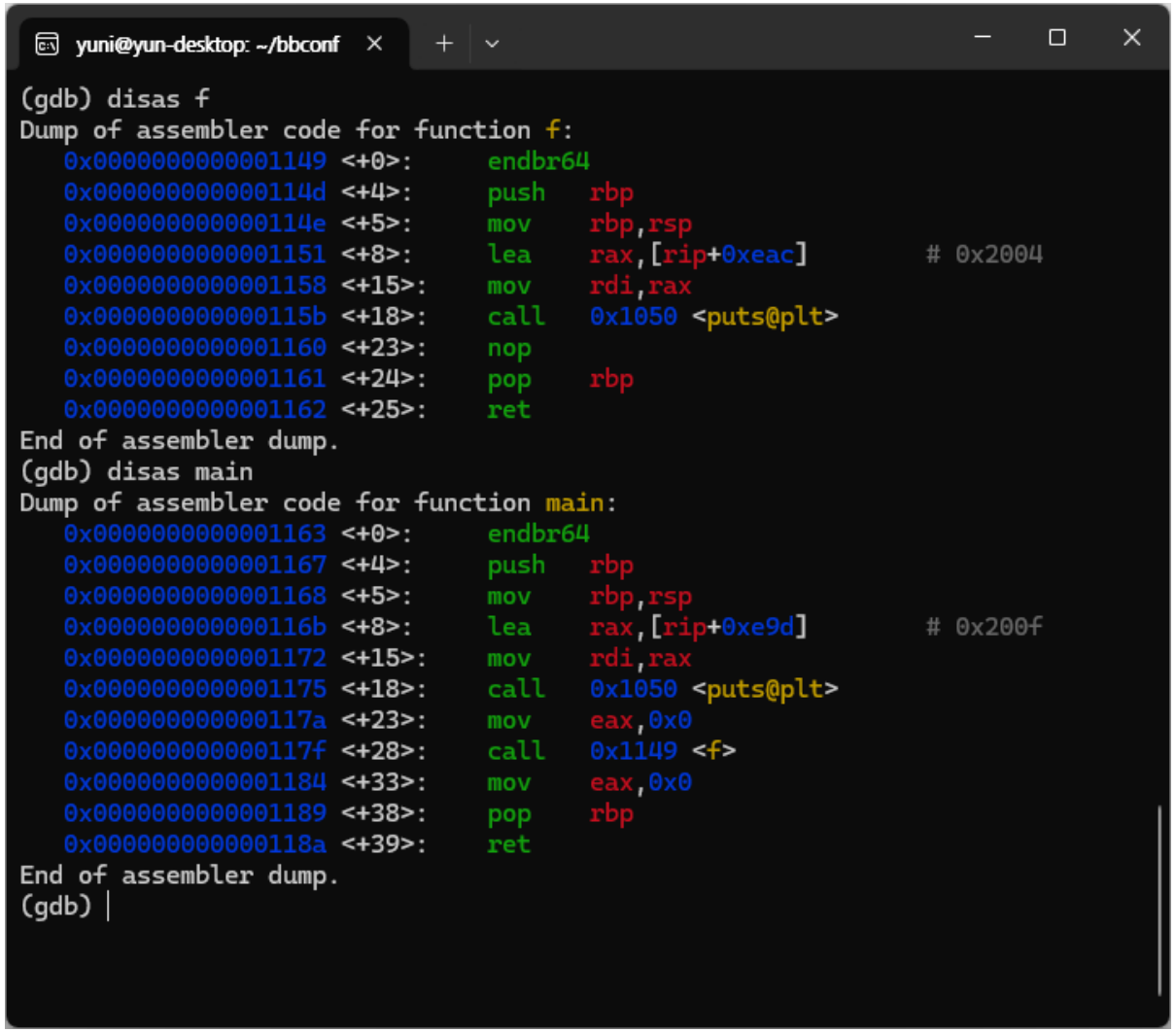
_start의 정체

```c
#include <stdio.h>

void f() {
    puts("f function");
}

int main() {
    puts("main function");
    f();

    return 0;
}
```

Best record **14328** Bytes

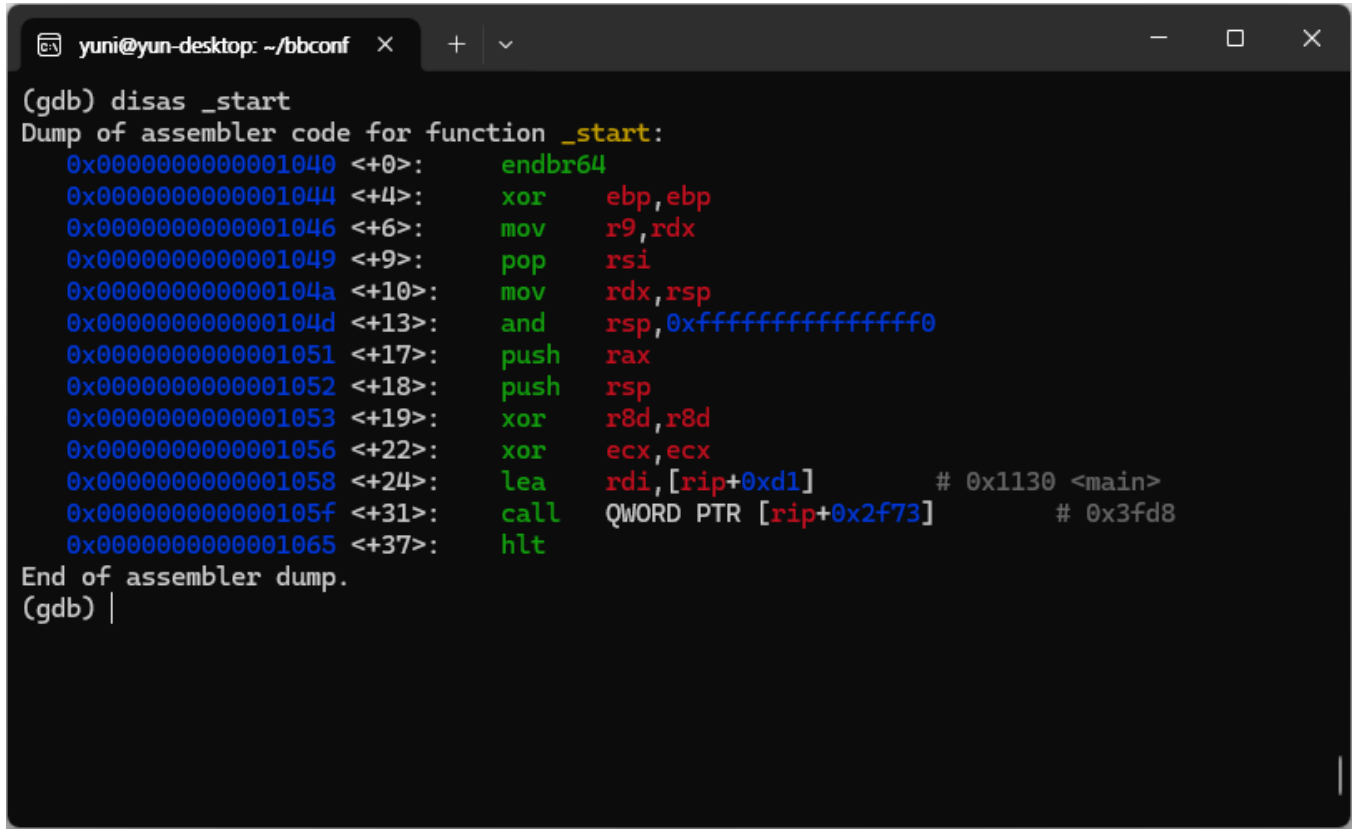
_start의 정체

```c
#include <stdio.h>

void f() {
    puts("f function");
}

int main() {
    puts("main function");
    f();

    return 0;
}
```



Best record **14328** Bytes

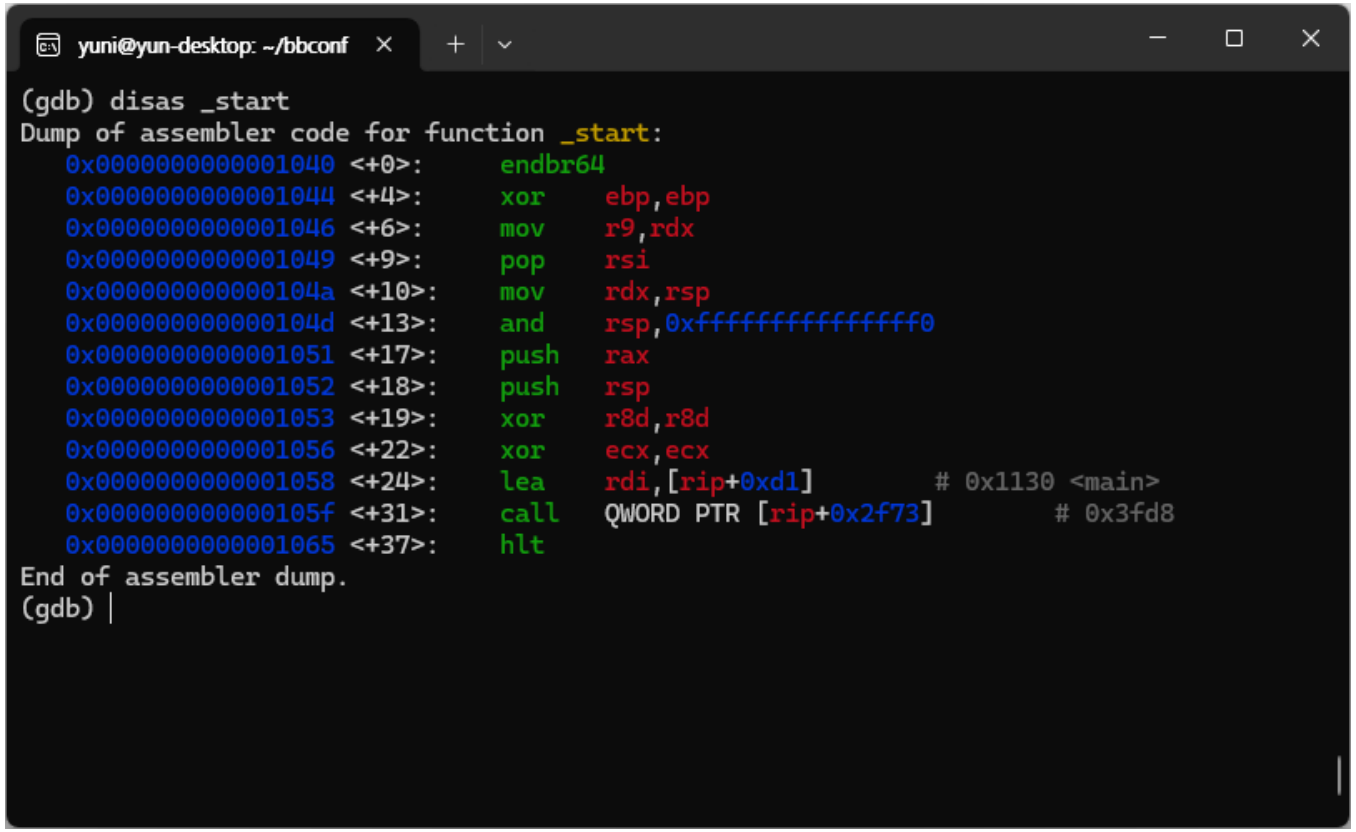
_start의 정체



다시 돌아와서...

Best record **14328** Bytes

_start의 정체



가만 살펴보니 함수는 아닌 듯함

Best record **14328** Bytes

_start의 정체

_start를 잘 종료시키는 방법 : **_exit()**

Best record **14328** Bytes

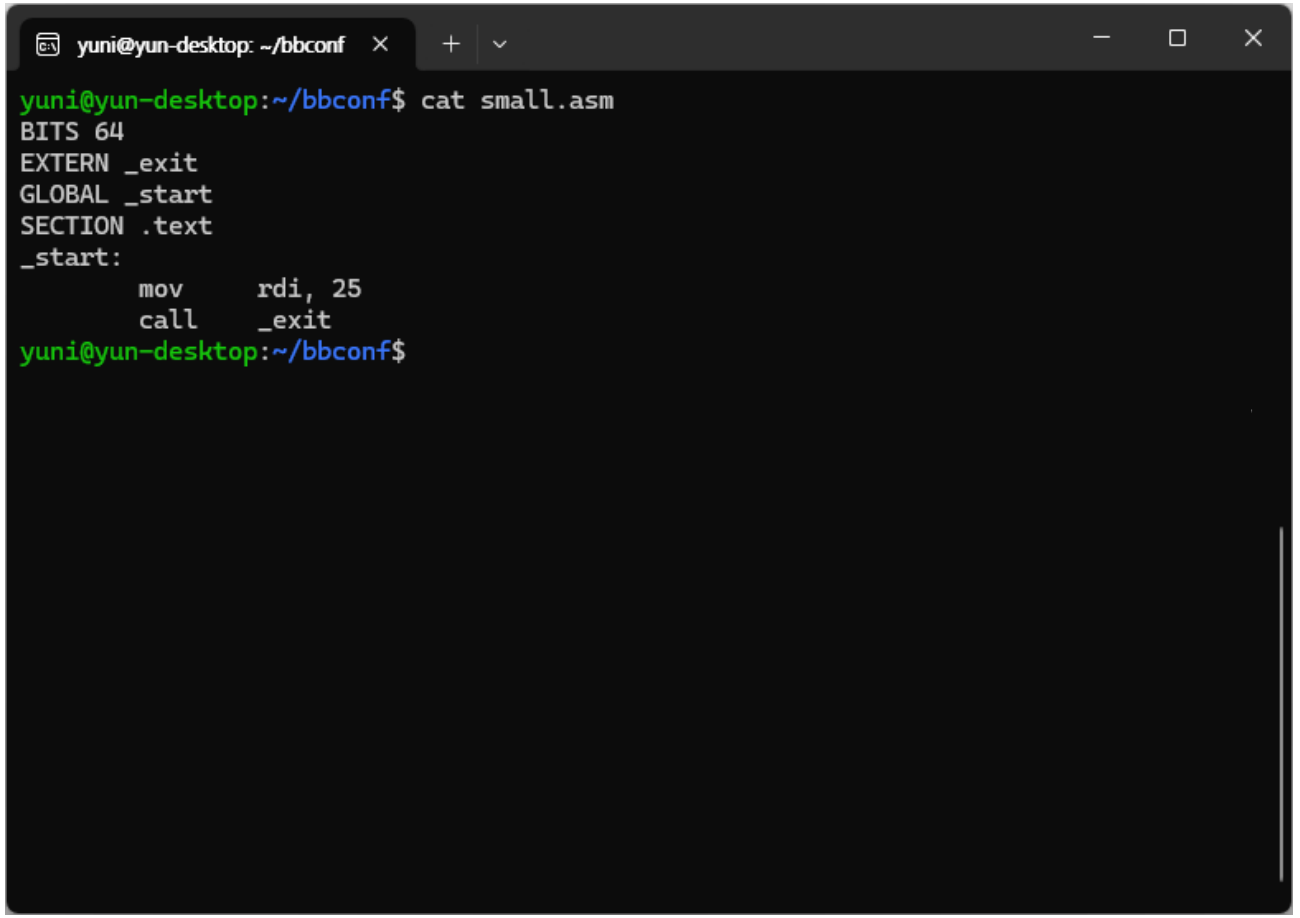
_start의 정체

```
BITS 64
EXTERN _exit
GLOBAL _start
SECTION .text
_start:
    mov     rdi, 25
    call    _exit
```

Best record **14328** Bytes

_start의 정체

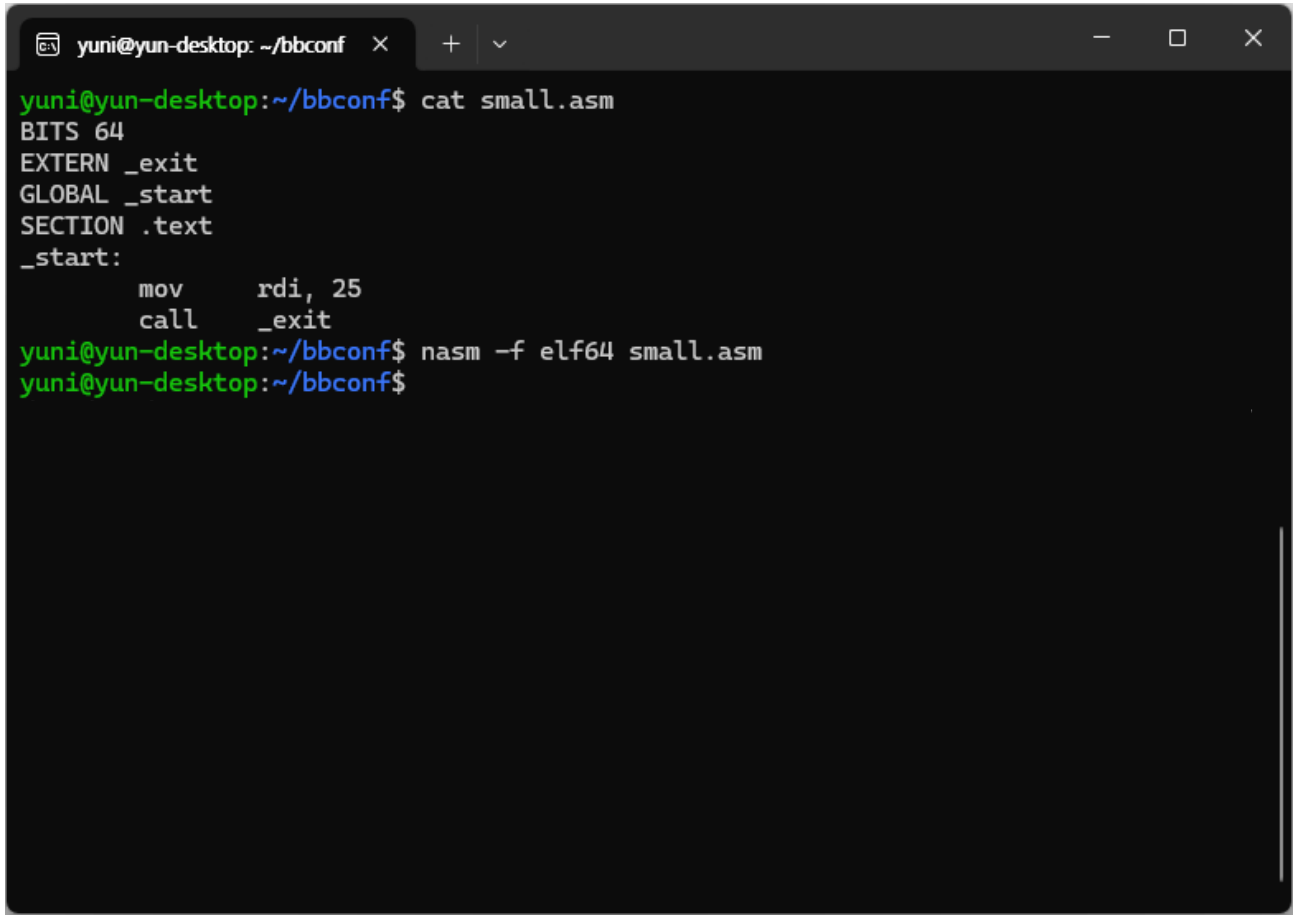
```
BITS 64
EXTERN _exit
GLOBAL _start
SECTION .text
_start:
    mov     rdi, 25
    call    _exit
```



Best record **14328** Bytes

_start의 정체

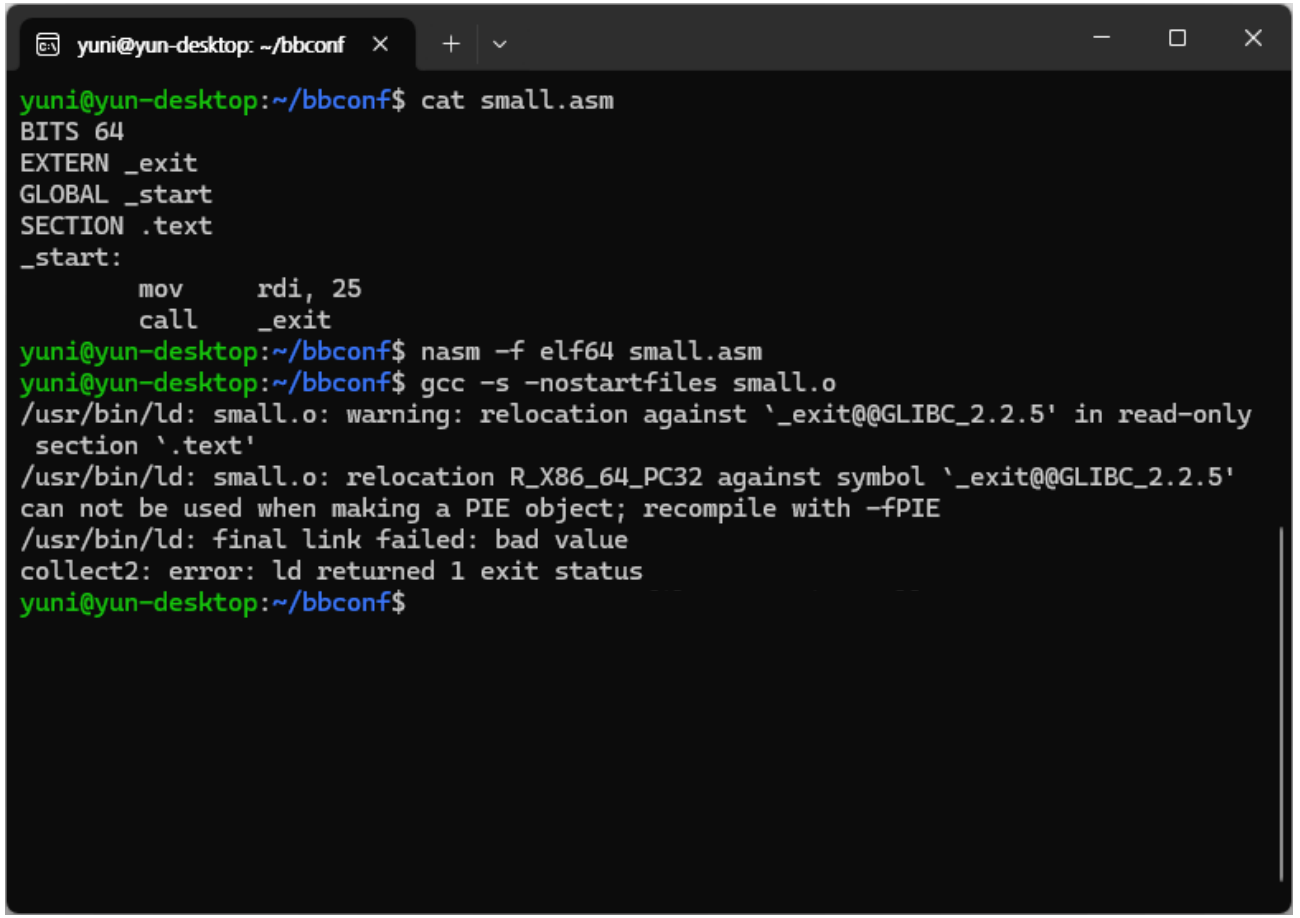
```
BITS 64
EXTERN _exit
GLOBAL _start
SECTION .text
_start:
    mov     rdi, 25
    call    _exit
```



Best record **14328** Bytes

_start의 정체

```
BITS 64
EXTERN _exit
GLOBAL _start
SECTION .text
_start:
    mov     rdi, 25
    call    _exit
```



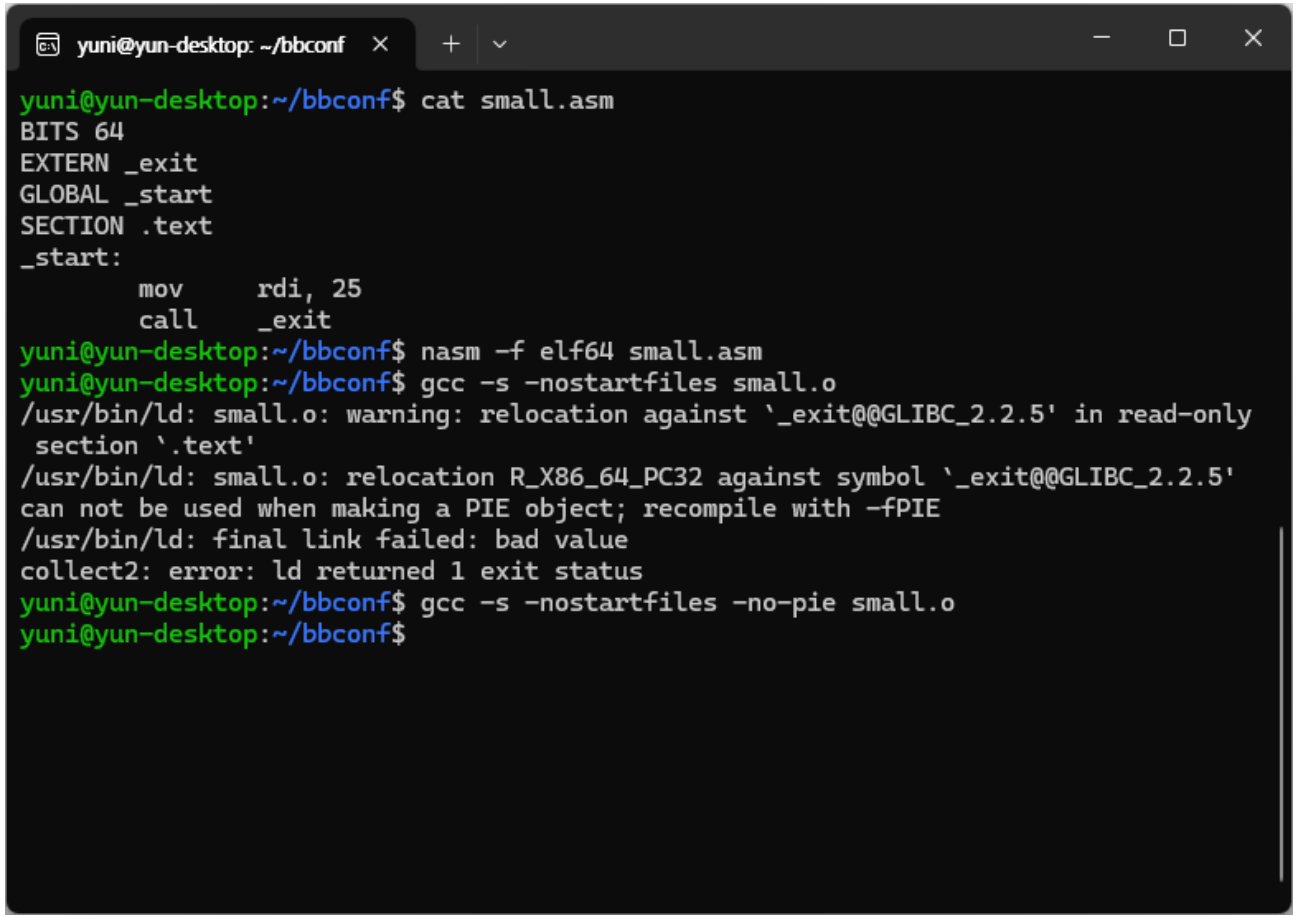
Best record **14328** Bytes

_start의 정체

PIE : Position Independent Executable

_start의 정체

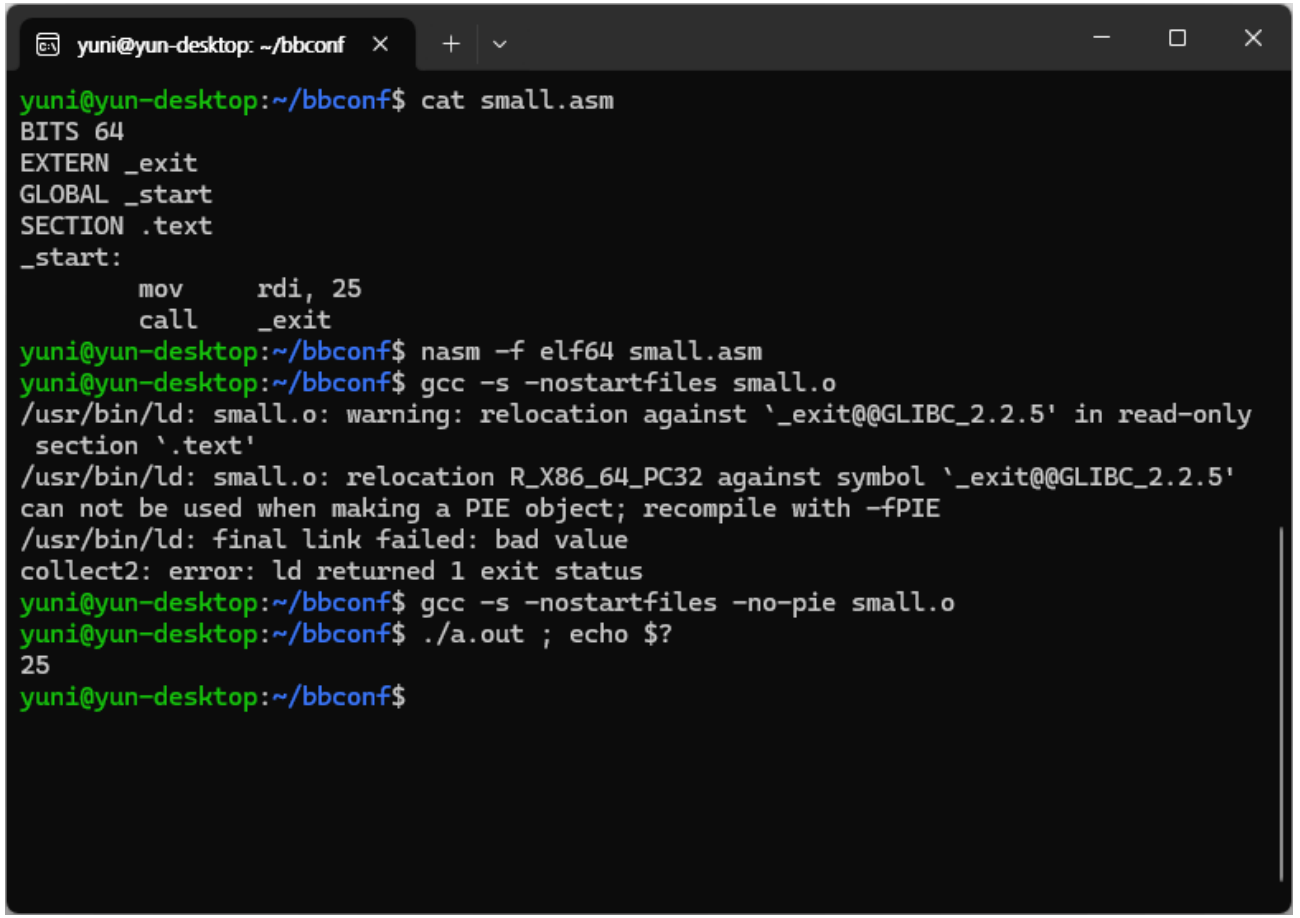
```
BITS 64
EXTERN _exit
GLOBAL _start
SECTION .text
_start:
    mov     rdi, 25
    call    _exit
```



Best record **14328** Bytes

_start의 정체

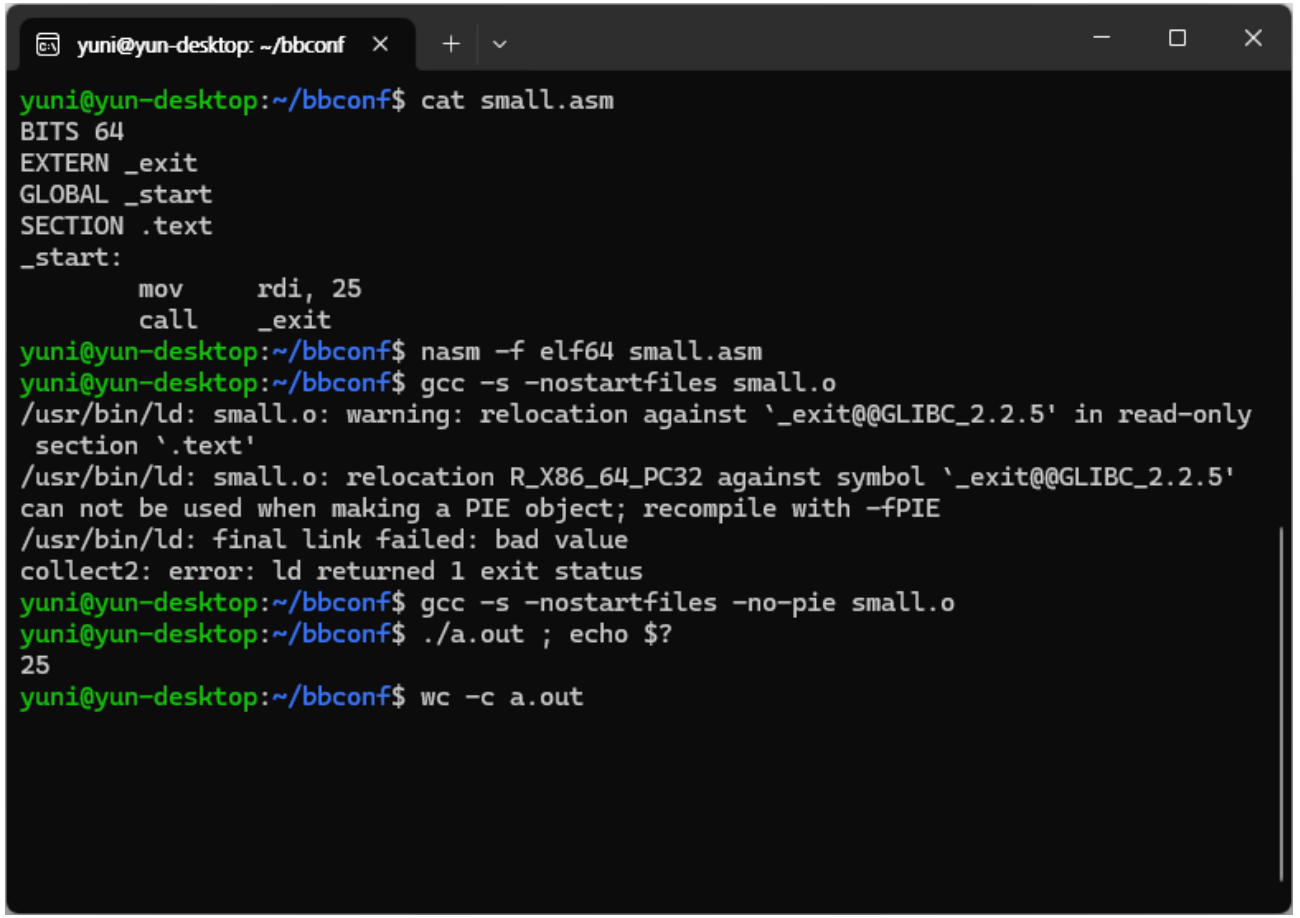
```
BITS 64
EXTERN _exit
GLOBAL _start
SECTION .text
_start:
    mov     rdi, 25
    call    _exit
```



Best record **14328** Bytes

_start의 정체

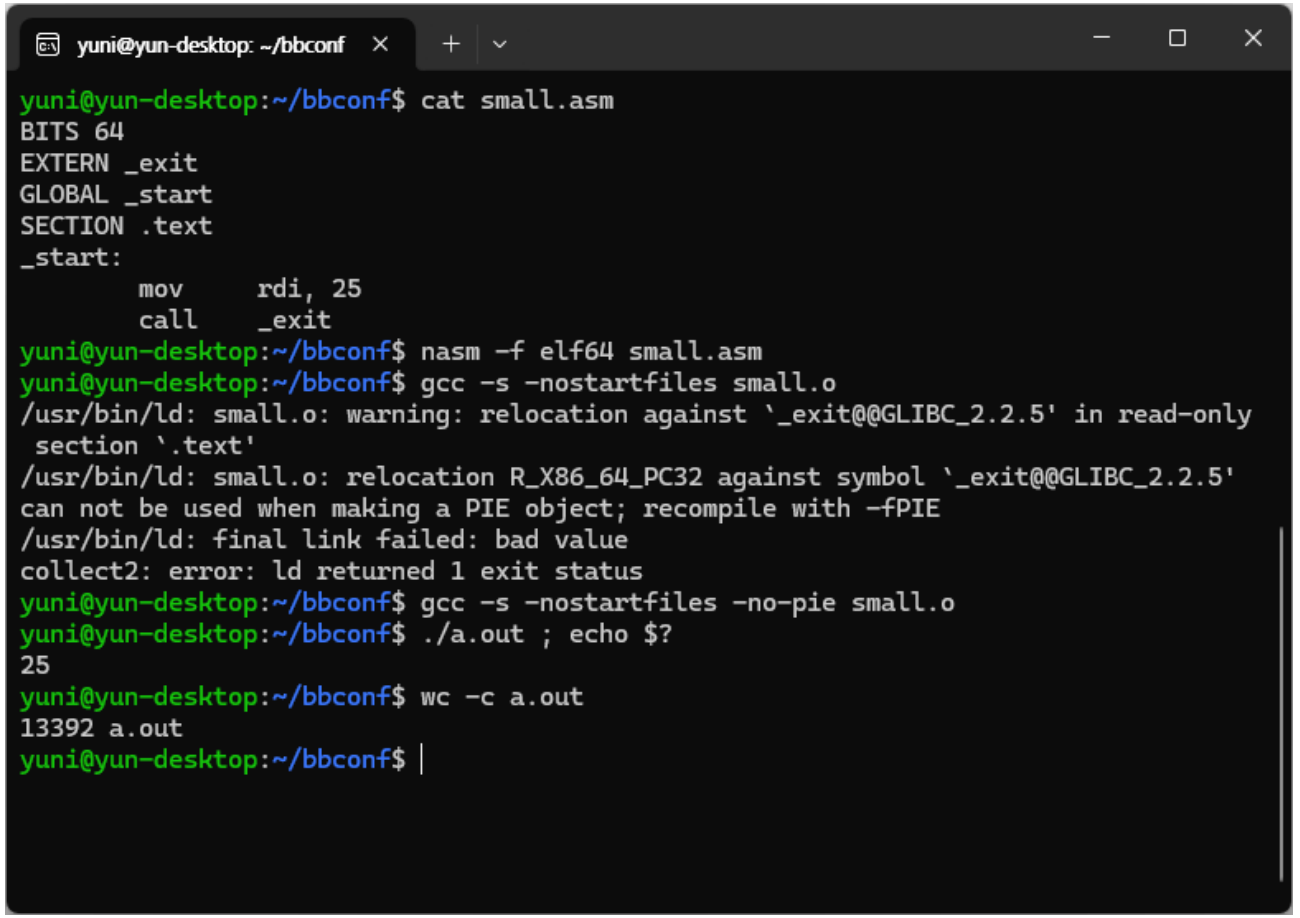
```
BITS 64
EXTERN _exit
GLOBAL _start
SECTION .text
_start:
    mov     rdi, 25
    call    _exit
```



Best record **14328** Bytes

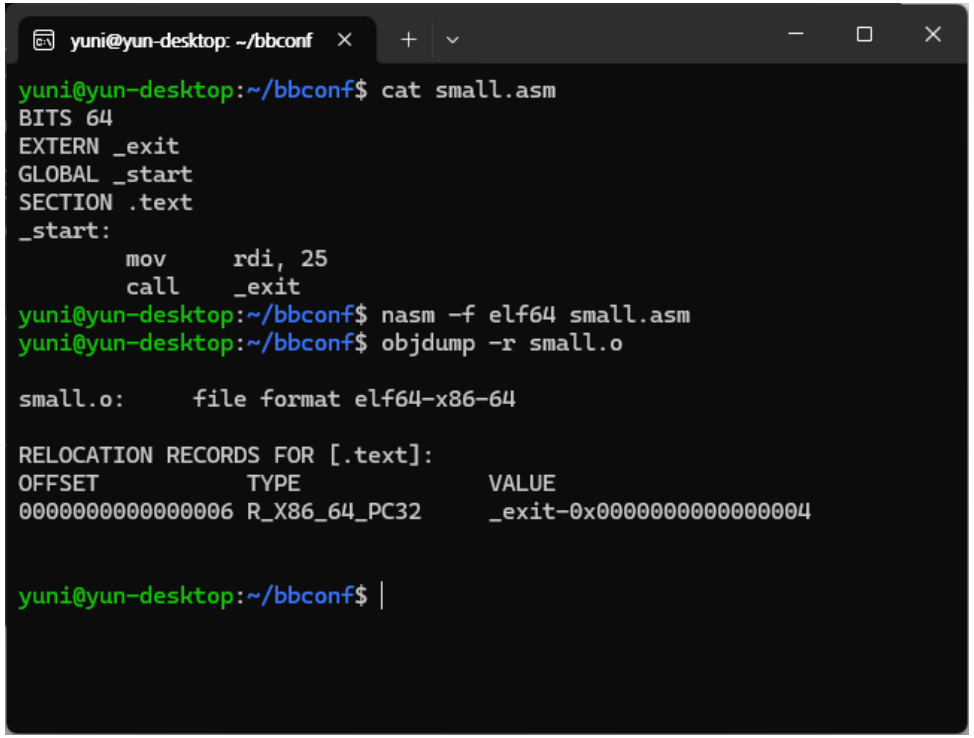
_start의 정체

```
BITS 64
EXTERN _exit
GLOBAL _start
SECTION .text
_start:
    mov     rdi, 25
    call    _exit
```



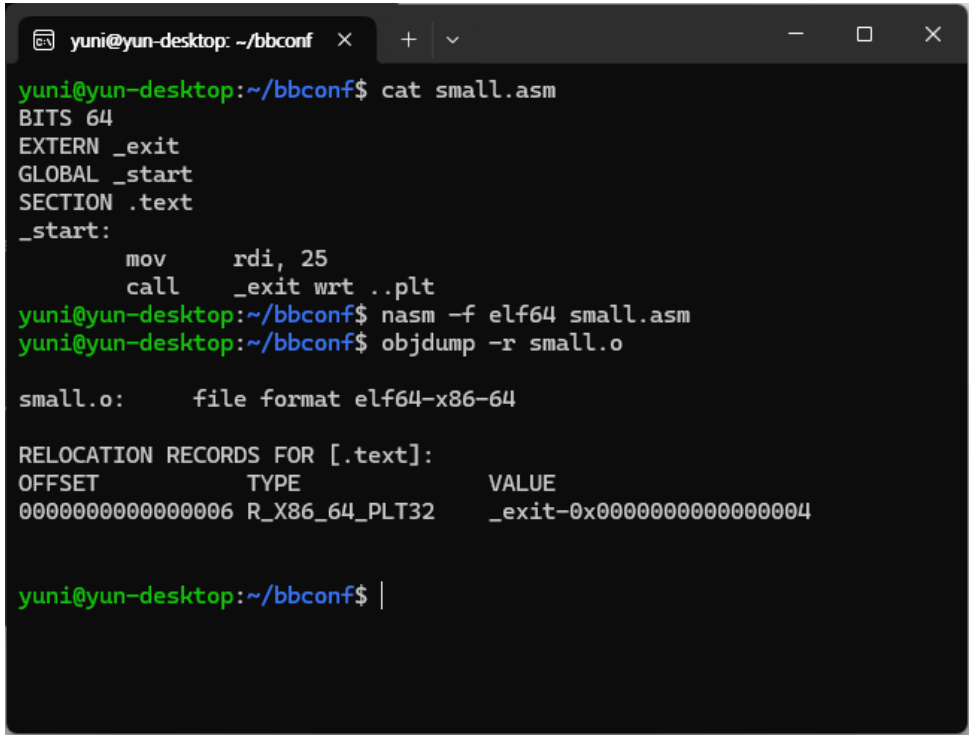
Best record **14328** -> **13392** Bytes (▼936)

_start의 정체



Best record **13392** Bytes

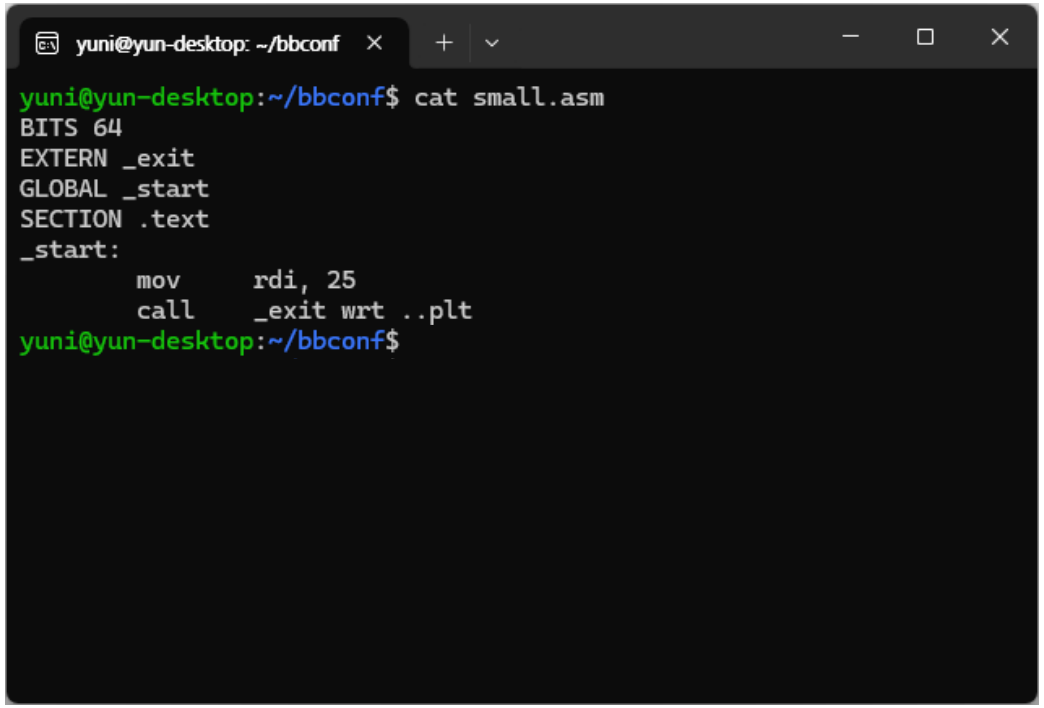
_start의 정체



Best record **13392** Bytes

_start의 정체

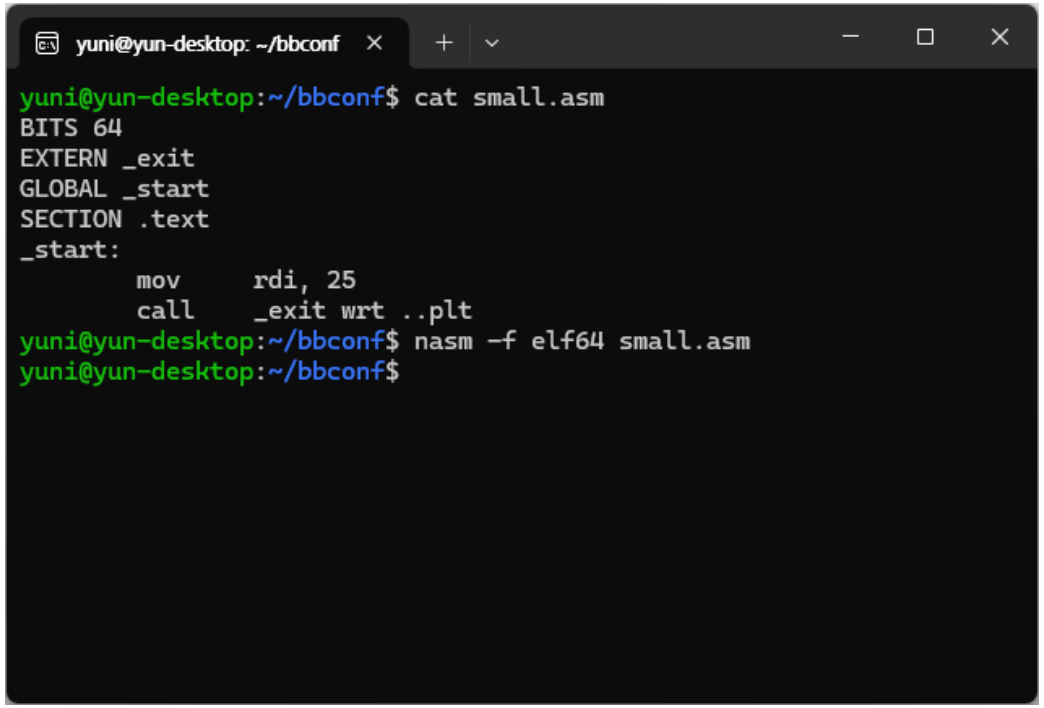
```
BITS 64
EXTERN _exit
GLOBAL _start
SECTION .text
_start:
    mov     rdi, 25
    call    _exit wrt ..plt
```



Best record **13392** Bytes

_start의 정체

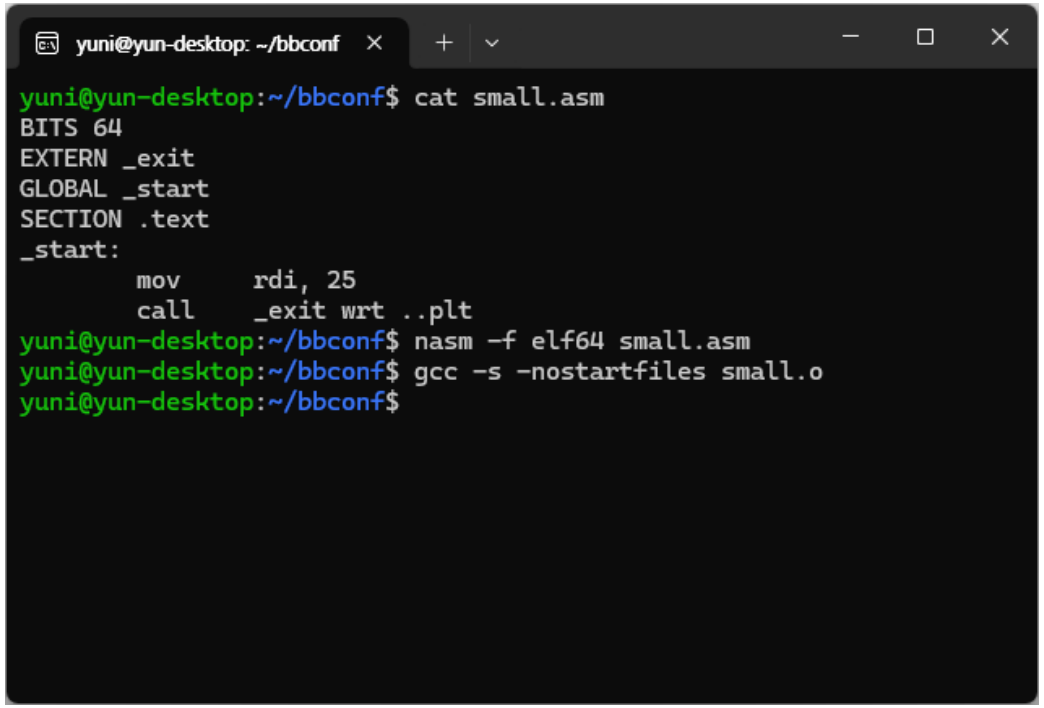
```
BITS 64
EXTERN _exit
GLOBAL _start
SECTION .text
_start:
    mov     rdi, 25
    call    _exit wrt ..plt
```



Best record **13392** Bytes

_start의 정체

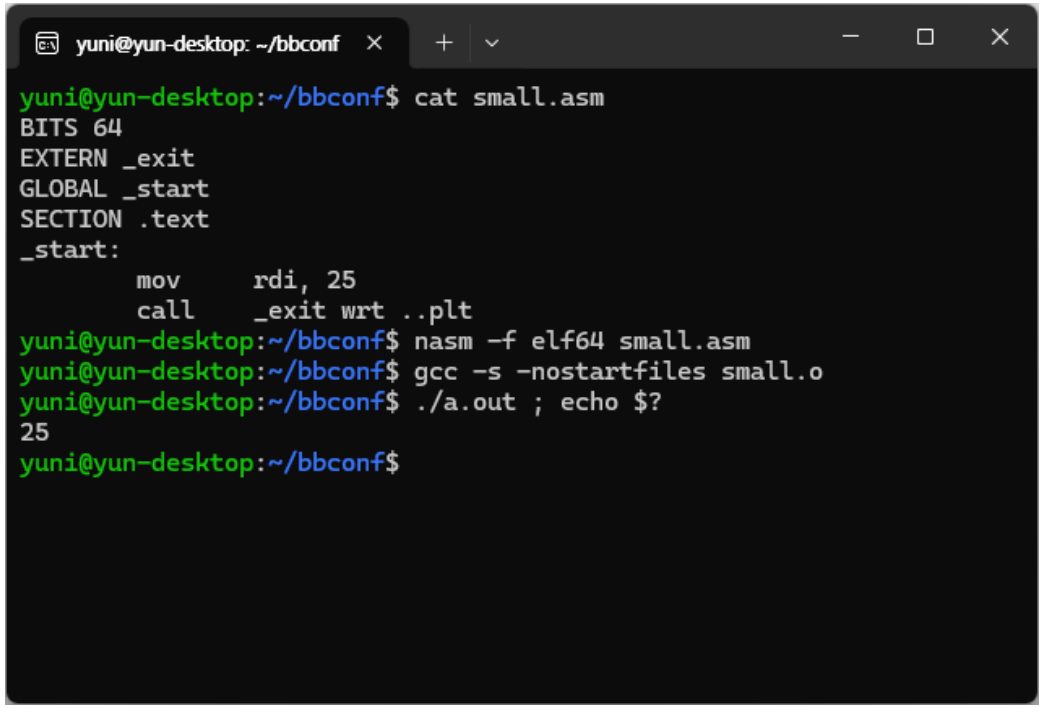
```
BITS 64
EXTERN _exit
GLOBAL _start
SECTION .text
_start:
    mov     rdi, 25
    call    _exit wrt ..plt
```



Best record **13392** Bytes

_start의 정체

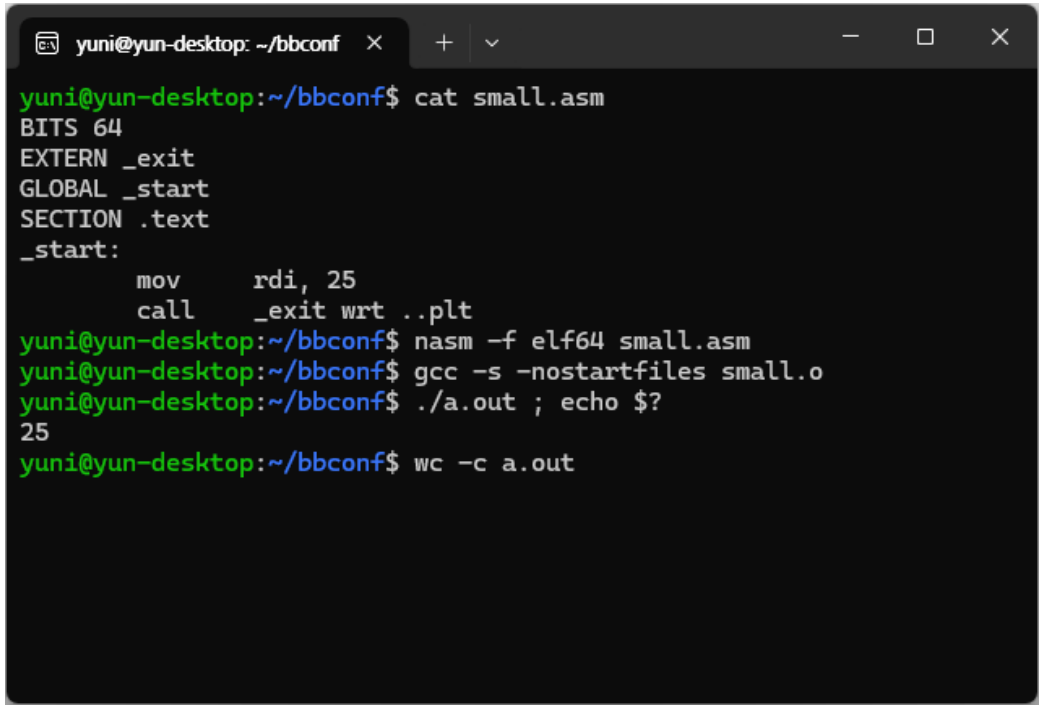
```
BITS 64
EXTERN _exit
GLOBAL _start
SECTION .text
_start:
    mov     rdi, 25
    call    _exit wrt ..plt
```



Best record **13392** Bytes

_start의 정체

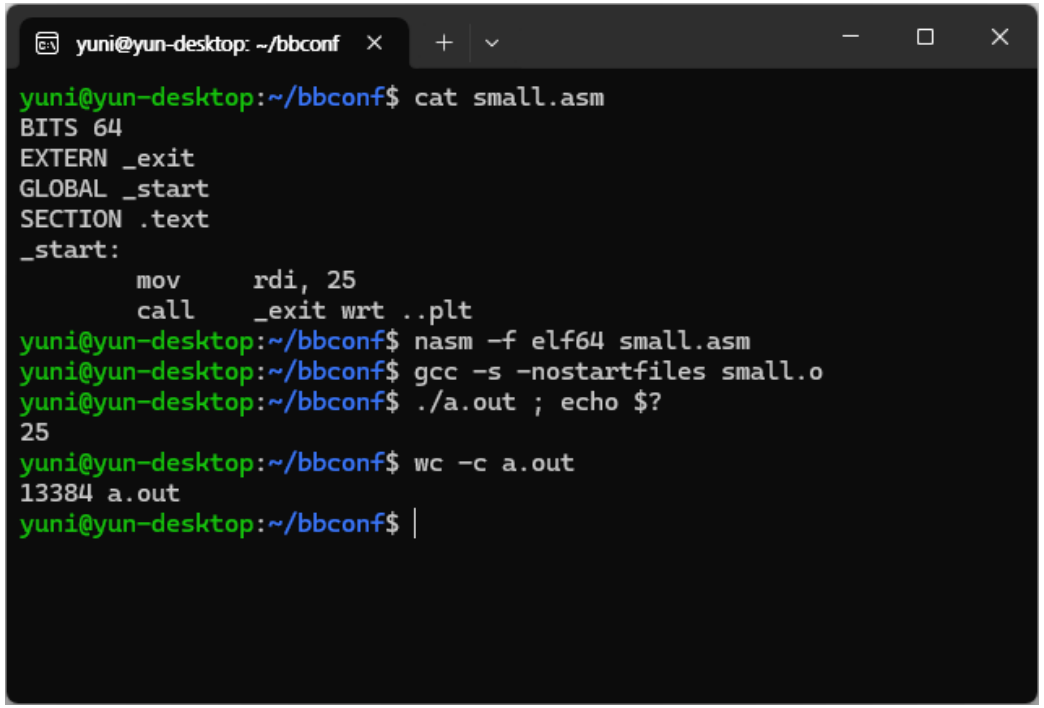
```
BITS 64
EXTERN _exit
GLOBAL _start
SECTION .text
_start:
    mov     rdi, 25
    call    _exit wrt ..plt
```



Best record **13392** Bytes

_start의 정체

```
BITS 64
EXTERN _exit
GLOBAL _start
SECTION .text
_start:
    mov     rdi, 25
    call    _exit wrt ..plt
```



Best record **13392** -> **13384** Bytes (▼8)

gcc –nostdlib 옵션

-nostdlib
Do not use the standard system startup files or libraries when linking. No startup files and only the libraries you specify are passed to the linker, and options specifying linkage of the system libraries, such as **-static-libgcc** or **-shared-libgcc**, are ignored.

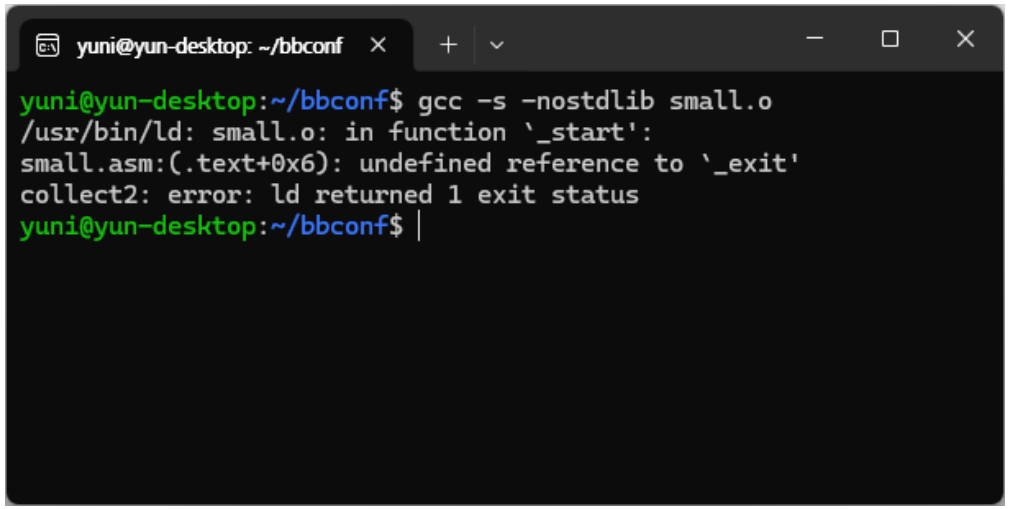
The compiler may generate calls to "memcmp", "memset", "memcpy" and "memmove". These entries are usually resolved by entries in libc. These entry points should be supplied through some other mechanism when this option is specified.

One of the standard libraries bypassed by **-nostdlib** and **-nodefaultlibs** is *libgcc.a*, a library of internal subroutines which GCC uses to overcome shortcomings of particular machines, or special needs for some languages.

In most cases, you need *libgcc.a* even when you want to avoid other standard libraries. In other words, when you specify **-nostdlib** or **-nodefaultlibs** you should usually specify **-lgcc** as well. This ensures that you have no unresolved references to internal GCC library subroutines. (An example of such an internal subroutine is "__main", used to ensure C++ constructors are called.)

Best record **13384** Bytes

gcc –nostdlib 옵션



Best record **13384** Bytes

직접 syscall 호출

glibc / sysdeps / unix / sysv / linux / _exit.c

jsm28 Update copyright dates with scripts/update-copyrights.

Code Blame 42 lines (35 loc) · 1.17 KB Code 55% faster with GitHub Copilot

```
1   /* Copyright (C) 2002-2018 Free Software Foundation, Inc.
2      This file is part of the GNU C Library.
3
4      The GNU C Library is free software; you can redistribute it and/or
5      modify it under the terms of the GNU Lesser General Public
6      License as published by the Free Software Foundation; either
7      version 2.1 of the License, or (at your option) any later version.
8
9      The GNU C Library is distributed in the hope that it will be useful,
10     but WITHOUT ANY WARRANTY; without even the implied warranty of
11     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
12     Lesser General Public License for more details.
13
14     You should have received a copy of the GNU Lesser General Public
15     License along with the GNU C Library; if not, see
16     <http://www.gnu.org/licenses/>.  */
17
18  #include <errno.h>
19  #include <stdlib.h>
20  #include <unistd.h>
21  #include <sysdep.h>
22  #include <abort-instr.h>
23
24
25  void
26  _exit (int status)
27  {
28    while (1)
29      {
30  #ifdef __NR_exit_group
31        INLINE_SYSCALL (exit_group, 1, status);
32  #endif
33        INLINE_SYSCALL (exit, 1, status);
34
35  #ifdef ABORT_INSTRUCTION
36        ABORT_INSTRUCTION;
37  #endif
38      }
39  }
40  libc_hidden_def (_exit)
41  rtld_hidden_def (_exit)
42  weak_alias (_exit, _Exit)
```

Best record **13384** Bytes

직접 syscall 호출

Tables

x86_64 (64-bit)

Compiled from Linux 4.14.0 headers.

NR	syscall name	references	%rax	arg0 (%rdi)	arg1 (%rsi)	arg2 (%rdx)
0	read	man/ cs/	0x00	unsigned int fd	char *buf	size_t count
1	write	man/ cs/	0x01	unsigned int fd	const char *buf	size_t count
2	open	man/ cs/	0x02	const char *filename	int flags	umode_t mode
3	close	man/ cs/	0x03	unsigned int fd	-	-
4	stat	man/ cs/	0x04	const char *filename	struct __old_kernel_stat *statbuf	-
60	exit	man/ cs/	0x3c	int error_code	-	-

Best record **13384** Bytes

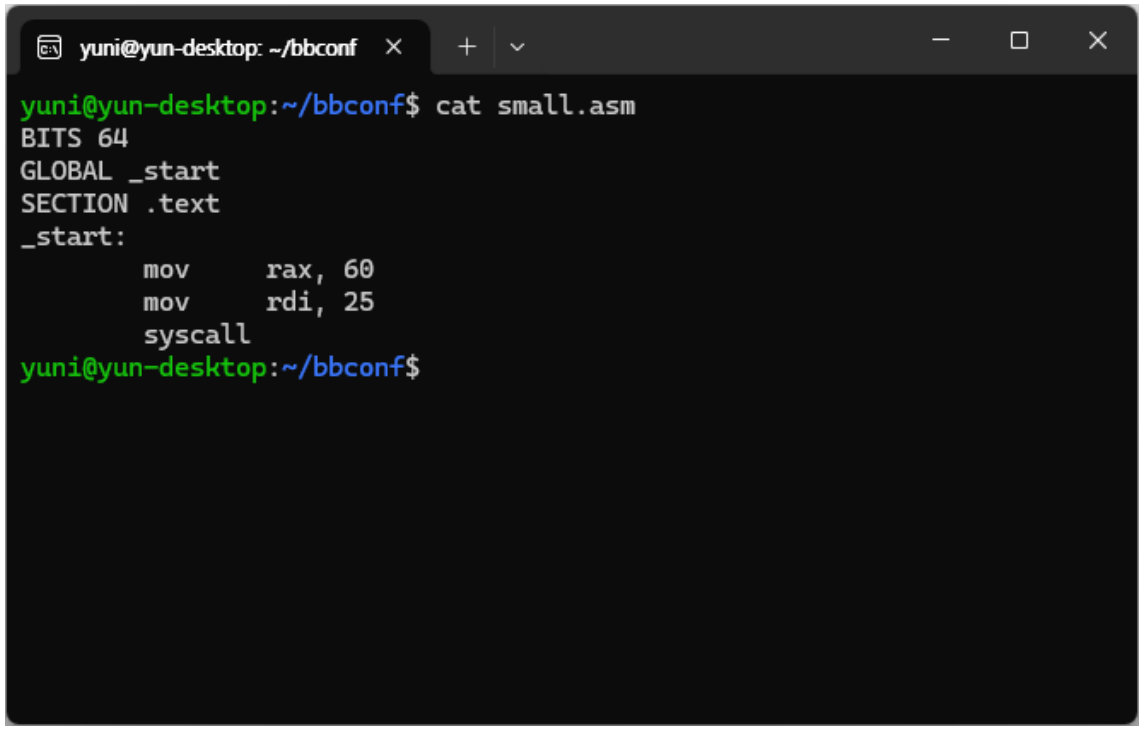
직접 syscall 호출

```
BITS 64
GLOBAL _start
SECTION .text
_start:
    mov     rax, 60
    mov     rdi, 25
    syscall
```

Best record **13384** Bytes

직접 syscall 호출

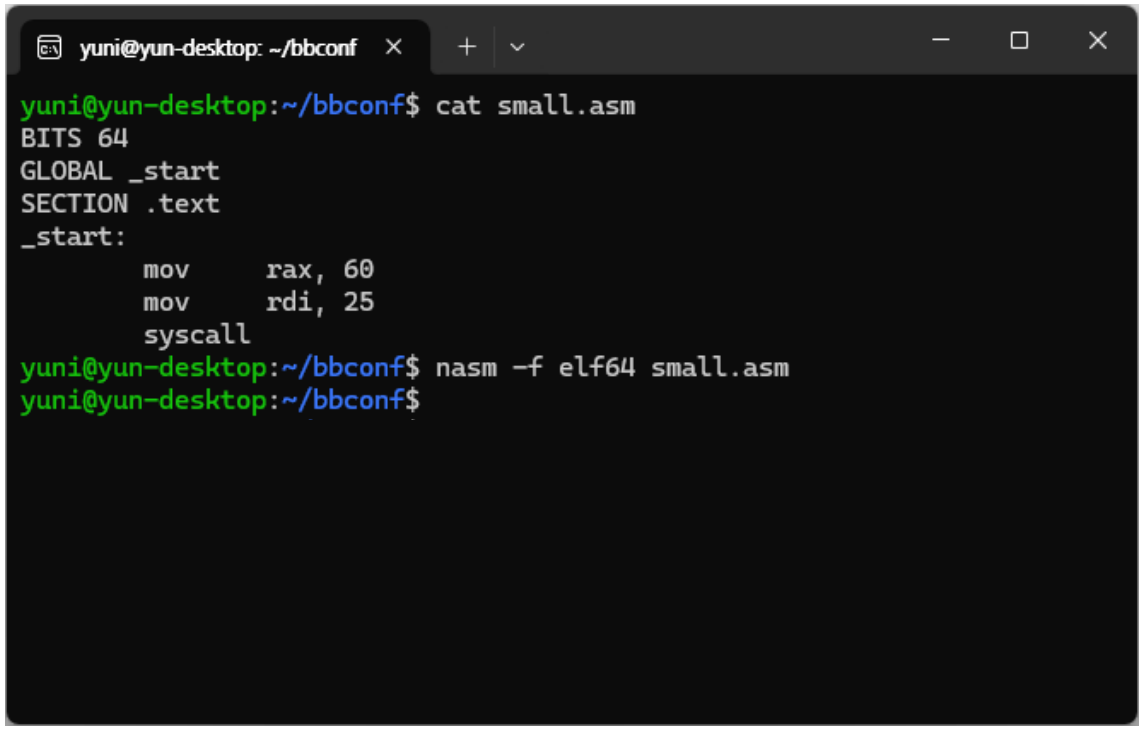
```
BITS 64
GLOBAL _start
SECTION .text
_start:
    mov     rax, 60
    mov     rdi, 25
    syscall
```



Best record **13384** Bytes

직접 syscall 호출

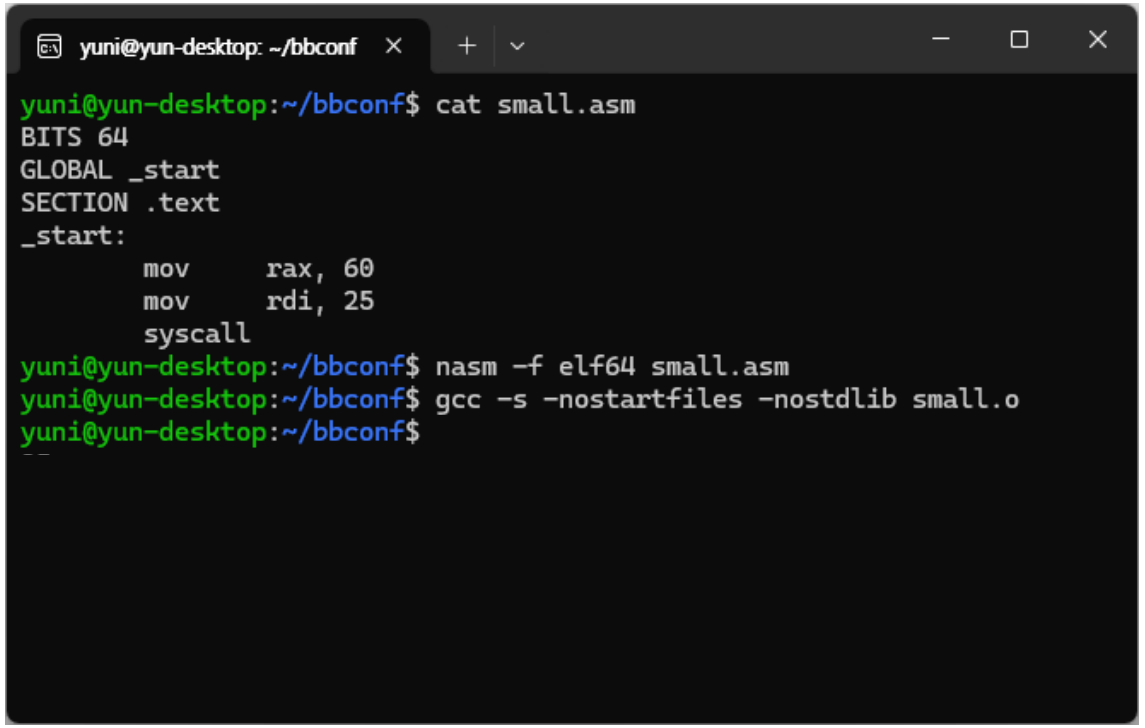
```
BITS 64
GLOBAL _start
SECTION .text
_start:
    mov     rax, 60
    mov     rdi, 25
    syscall
```



Best record **13384** Bytes

직접 syscall 호출

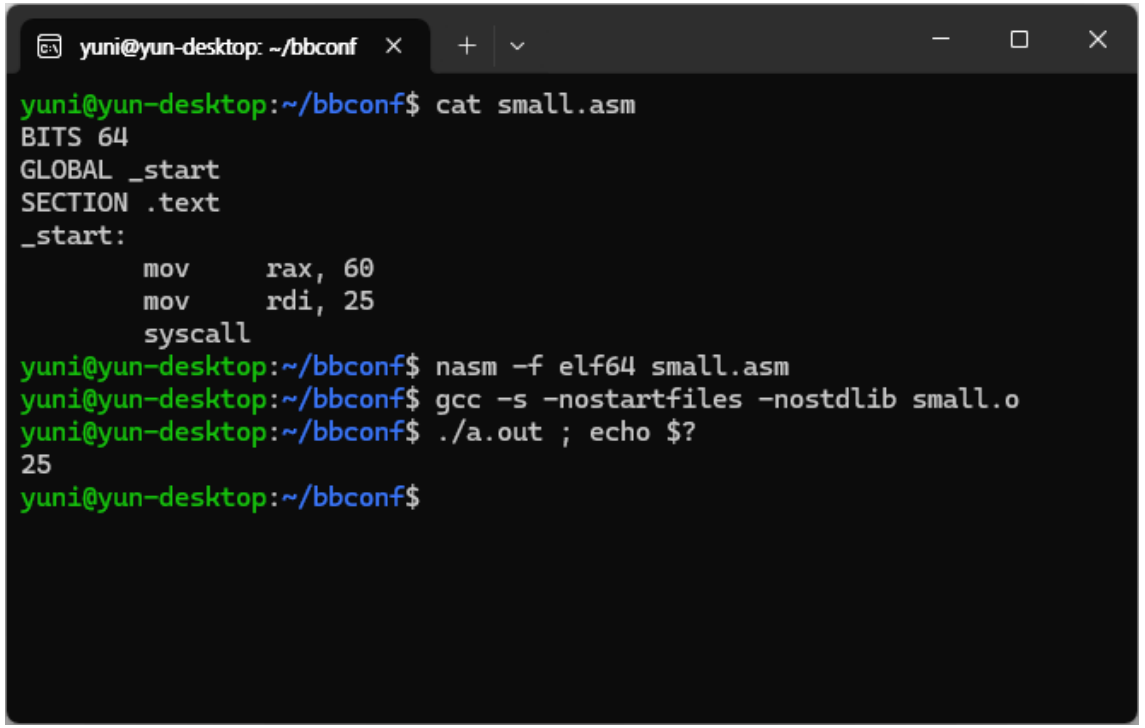
```
BITS 64
GLOBAL _start
SECTION .text
_start:
    mov     rax, 60
    mov     rdi, 25
    syscall
```



Best record **13384** Bytes

직접 syscall 호출

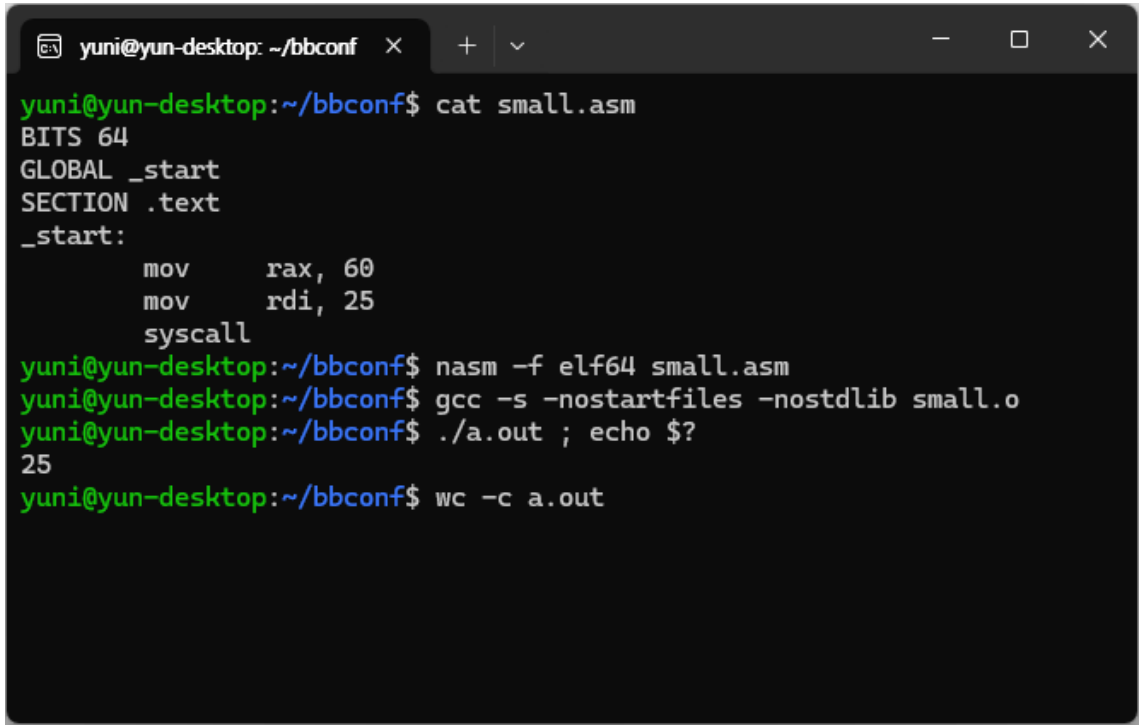
```
BITS 64
GLOBAL _start
SECTION .text
_start:
    mov     rax, 60
    mov     rdi, 25
    syscall
```



Best record **13384** Bytes

직접 syscall 호출

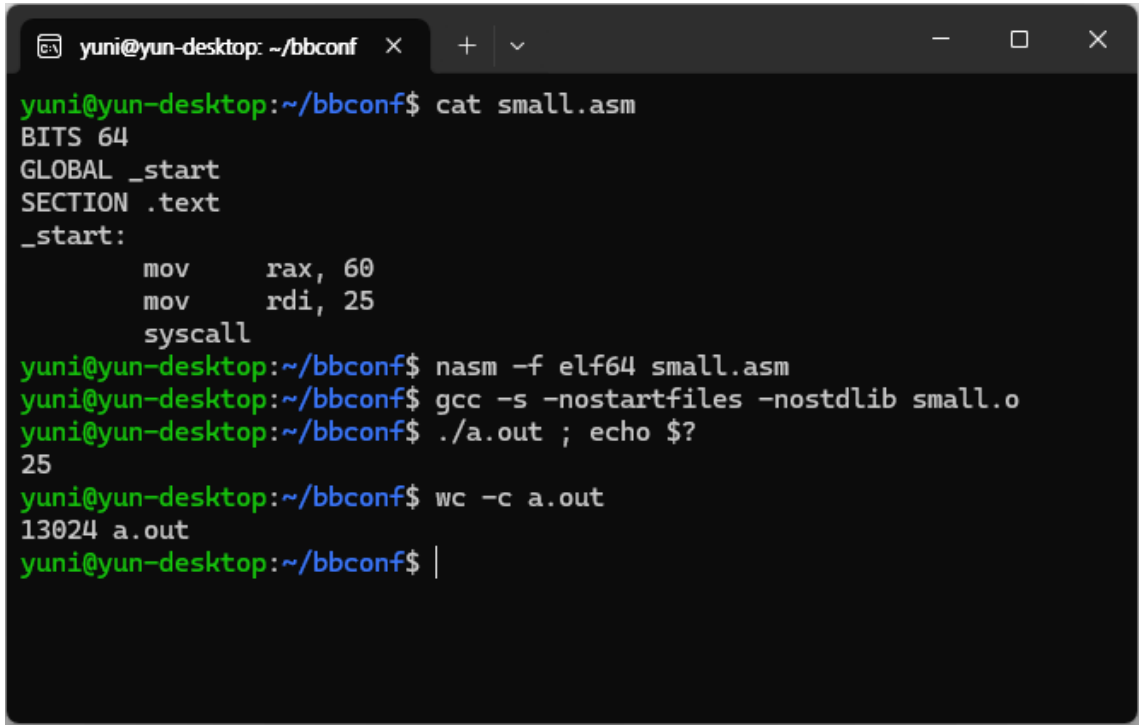
```
BITS 64
GLOBAL _start
SECTION .text
_start:
    mov     rax, 60
    mov     rdi, 25
    syscall
```



Best record **13384** Bytes

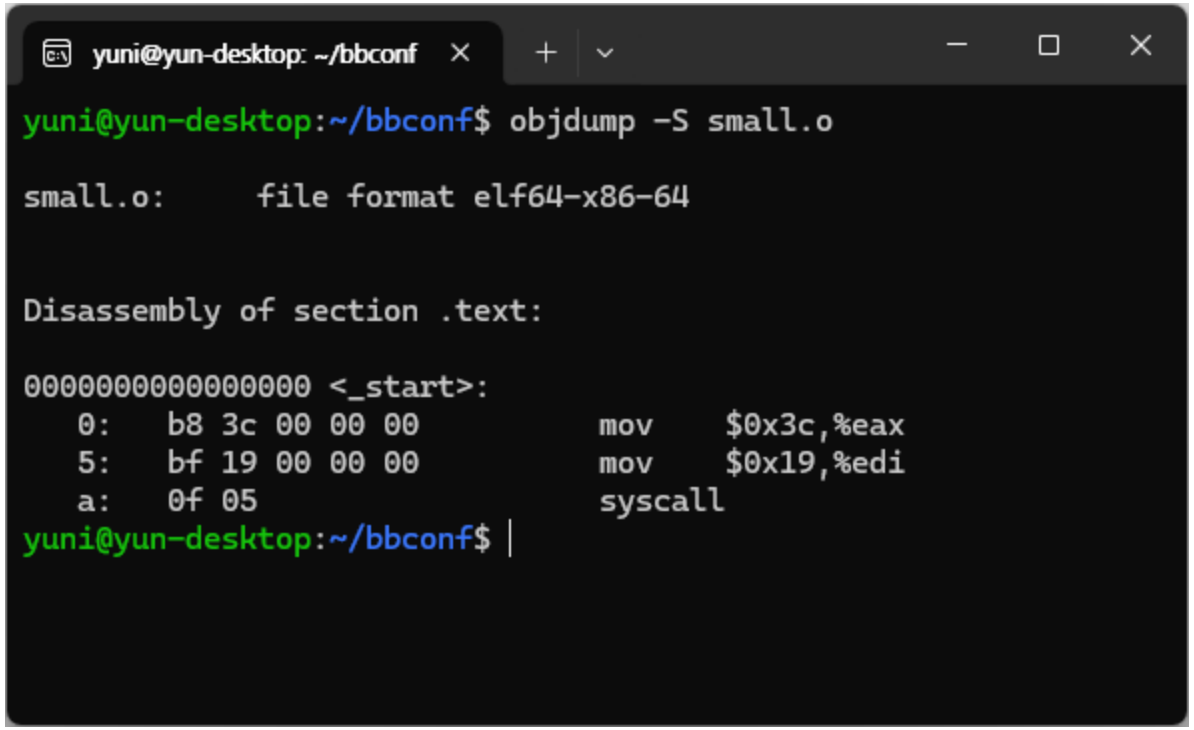
직접 syscall 호출

```
BITS 64
GLOBAL _start
SECTION .text
_start:
    mov     rax, 60
    mov     rdi, 25
    syscall
```



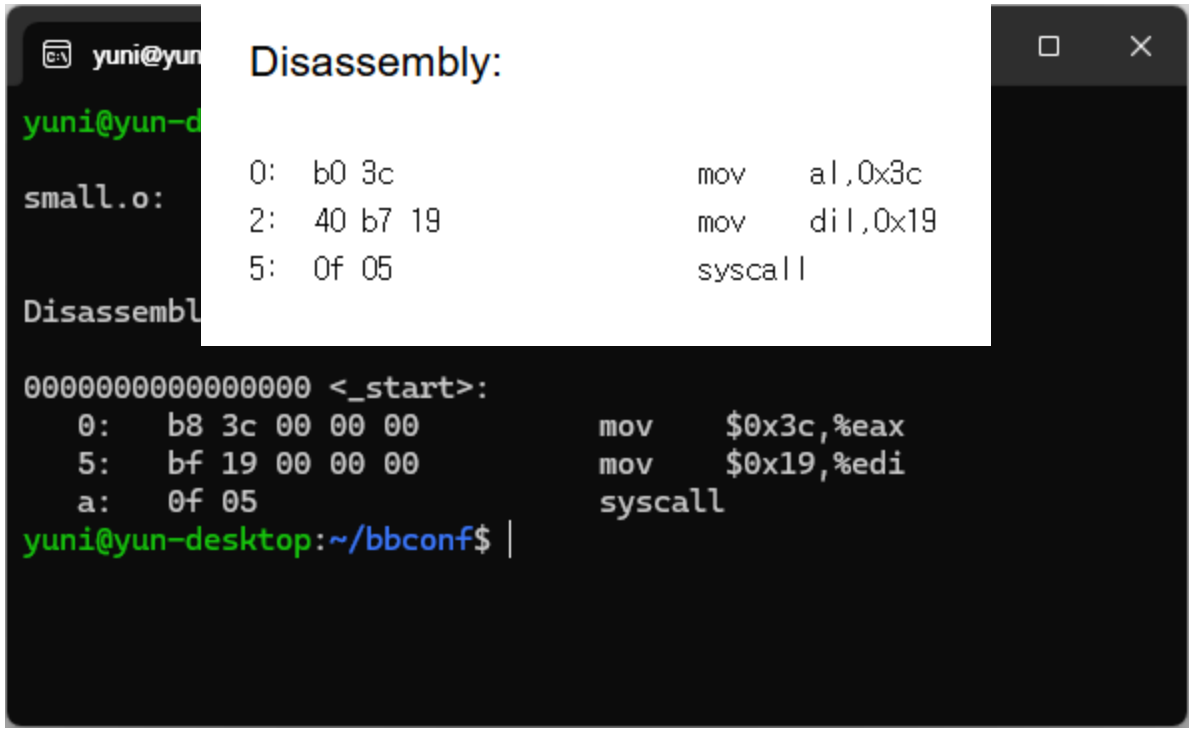
Best record **13384** -> **13024** Bytes (▼360)

어셈블리 쇼코딩



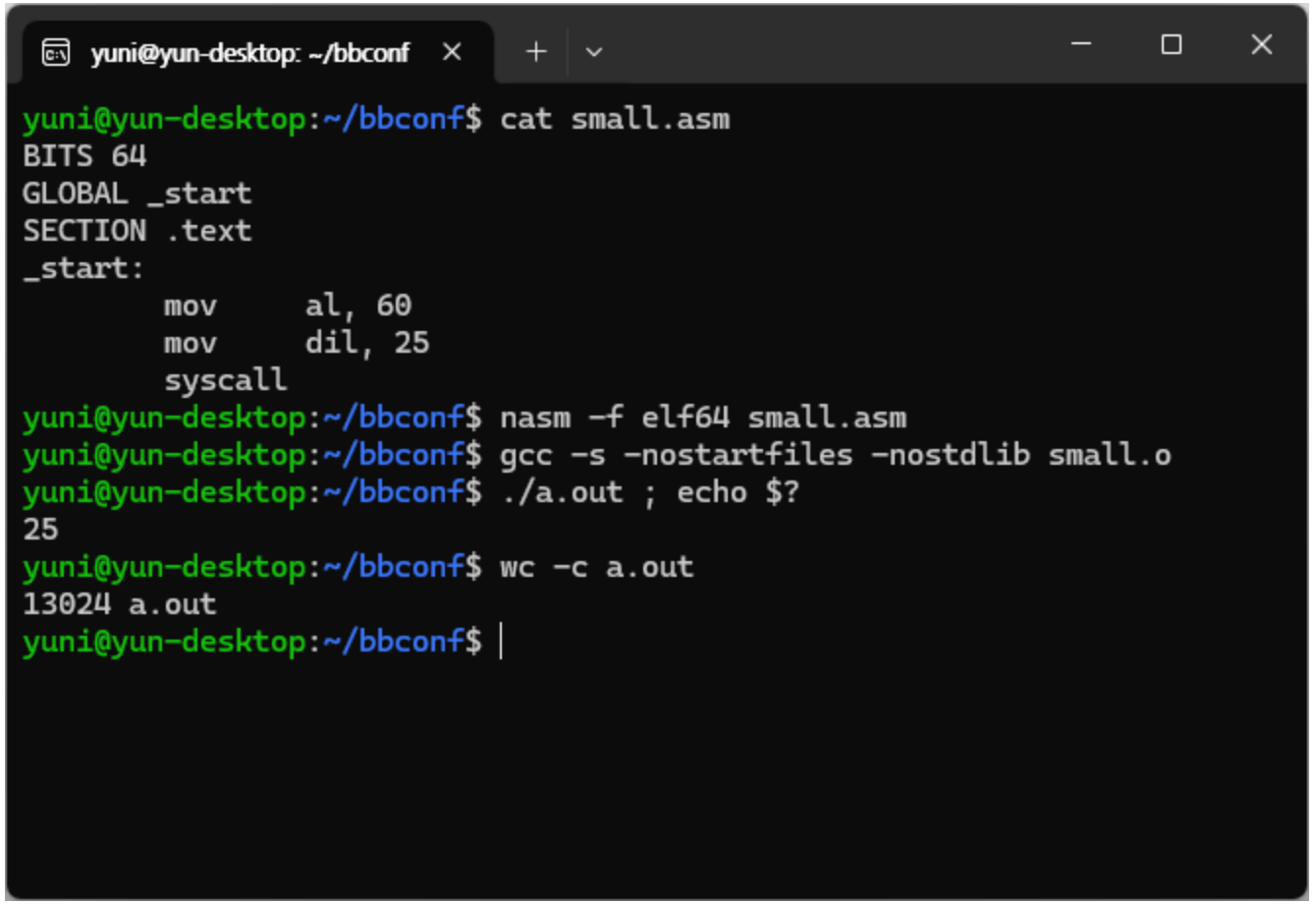
Best record **13024** Bytes

어셈블리 쇼코딩



Best record **13024** Bytes

어셈블리 쇼코딩



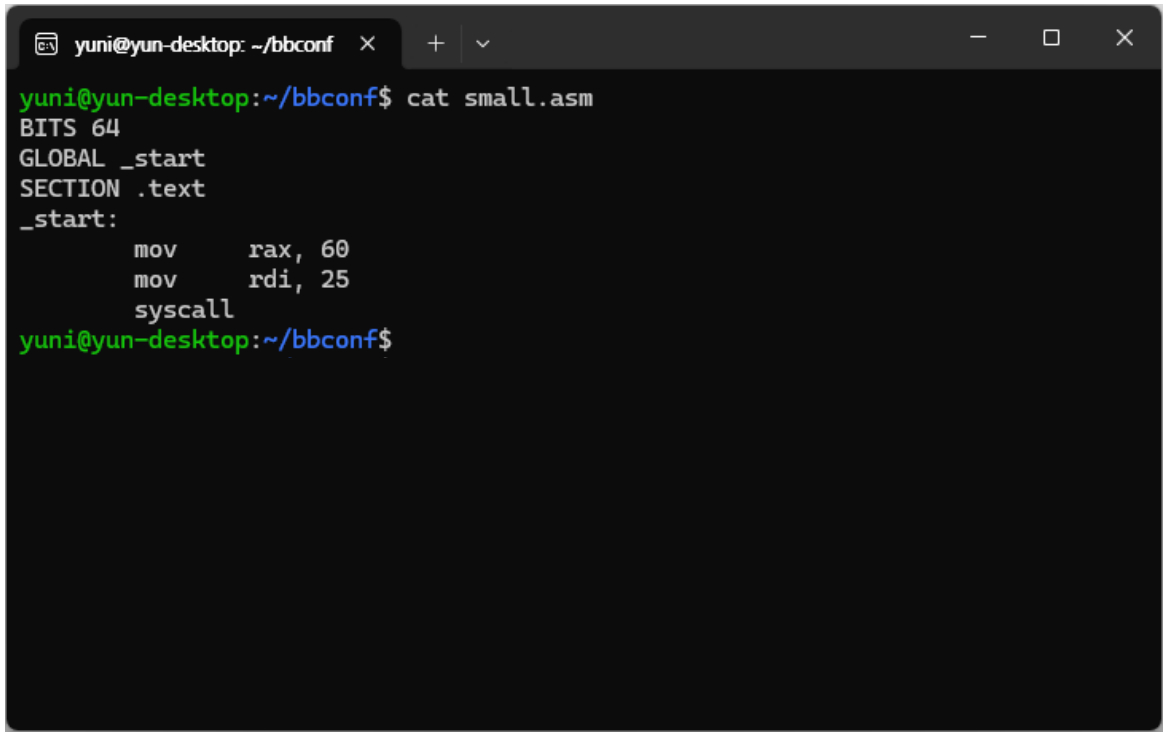
Best record **13024** Bytes

gcc –static 옵션

```
-static
    On systems that support dynamic linking, this overrides -pie
    and prevents linking with the shared libraries.  On other
    systems, this option has no effect.
```

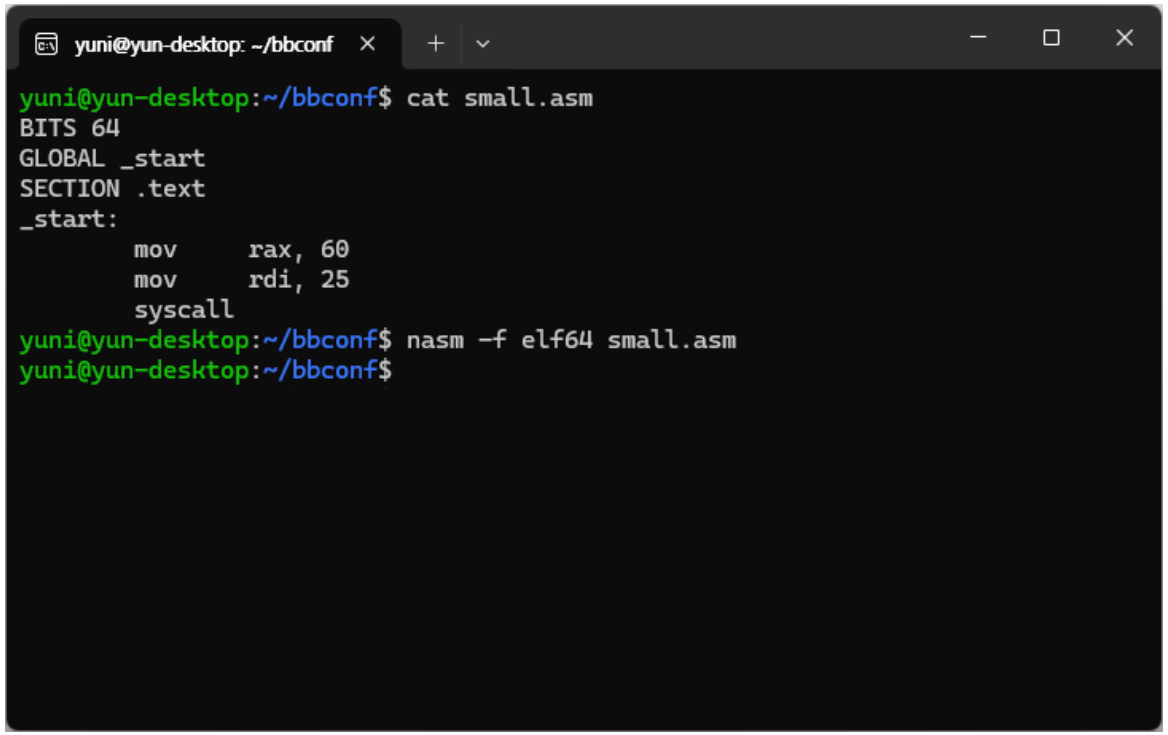
Best record **13024** Bytes

gcc –static 옵션



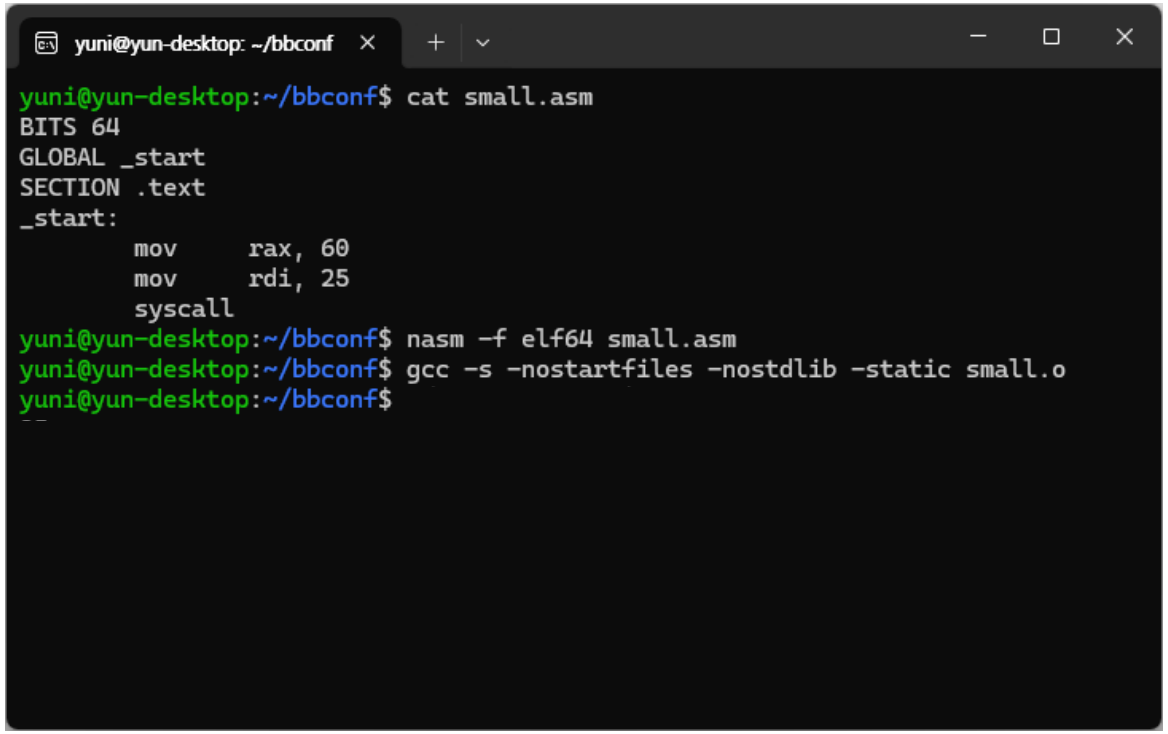
Best record **13024** Bytes

gcc –static 옵션



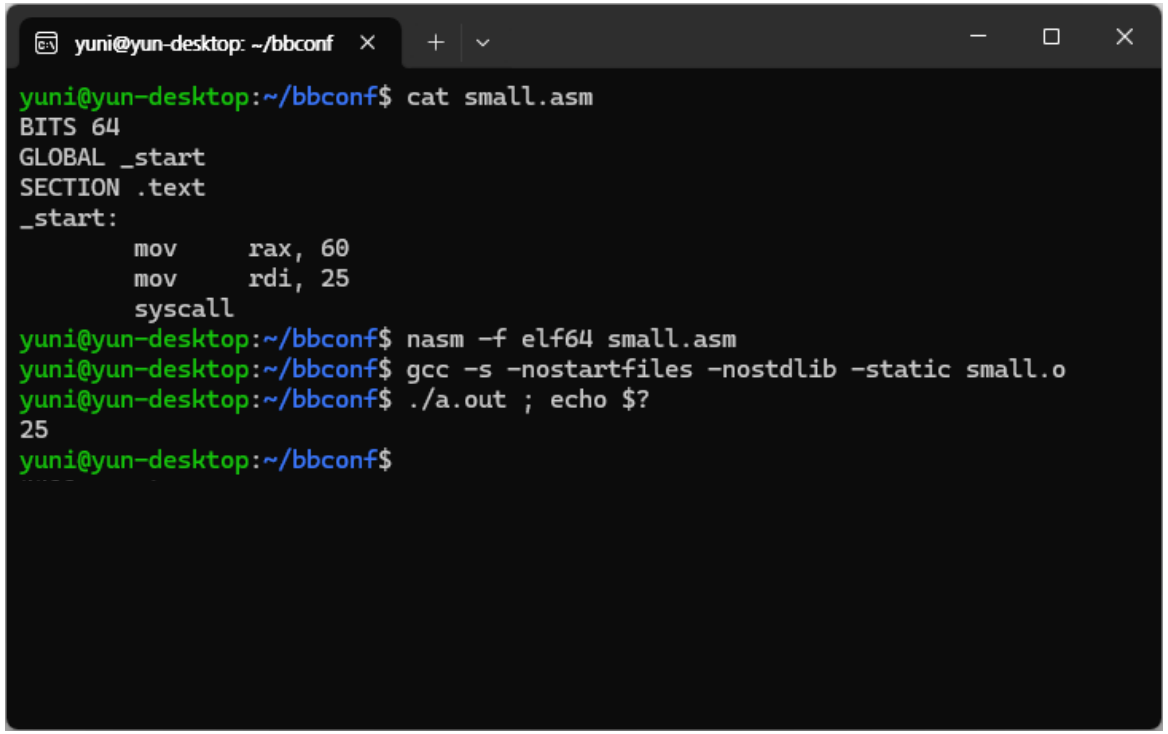
Best record **13024** Bytes

gcc –static 옵션



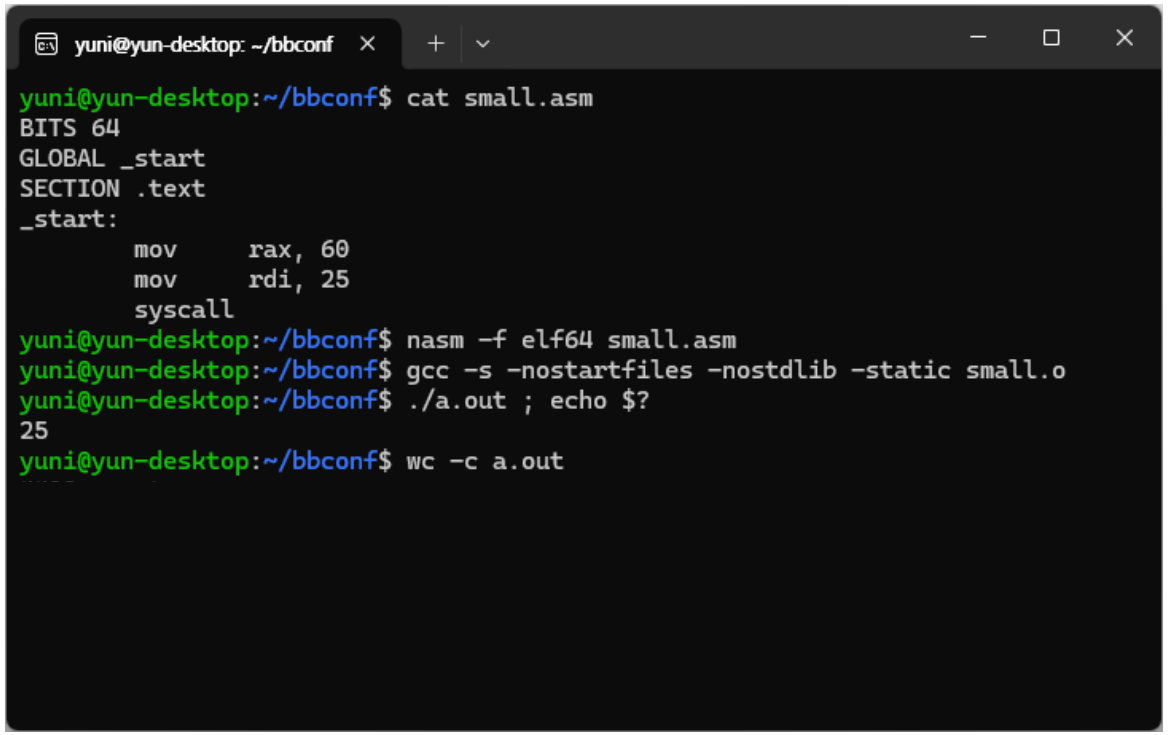
Best record **13024** Bytes

gcc –static 옵션



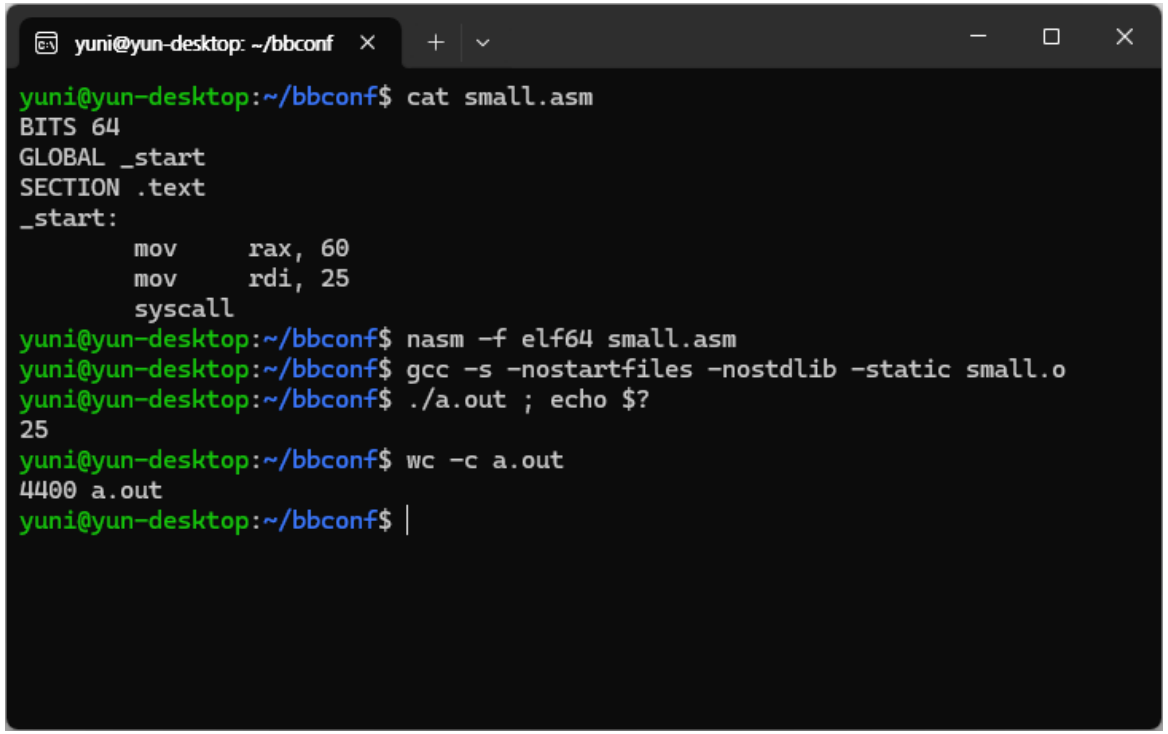
Best record **13024** Bytes

gcc –static 옵션



Best record **13024** Bytes

gcc –static 옵션



Best record **13024** -> **4400** Bytes (▼8624)

gcc는 버리고 ld로 가자

GNU 링커

9개 언어

문서 토론 읽기 편집 역사 보기 도구

위키백과, 우리 모두의 백과사전.

GNU 링커 (또는 **GNU ld**)는 유닉스 명령어 ld에 대한 GNU 프로젝트의 구현이다. GNU ld는 소프트웨어의 컴파일 시에 생성되는 목적 파일들로부터 실행 파일(또는 라이브러리)를 생성하는 링커를 실행한다. 링킹 프로세스에 대한 더 많은 권한을 행사하기 위해 **링커 스크립트**는 GNU ld로 전달된다.[1] GNU 링커는 GNU 바이너리 유틸리티(binutils)의 한 부분이다.

이름 "ld"의 기원으로 여겨지는 것들로는 "LoaD" 그리고 "Link eDitor"가 있다.[2]

GNU 링커는 자유 소프트웨어이며 GNU 일반 공중 사용 허가서 하에 배포된다.

GNU 링커

원저자	GNU 프로젝트
개발자	GNU 프로젝트
운영 체제	GNU
종류	링커
라이선스	GNU 일반 공중 사용 허가서

같이 보기 [편집]

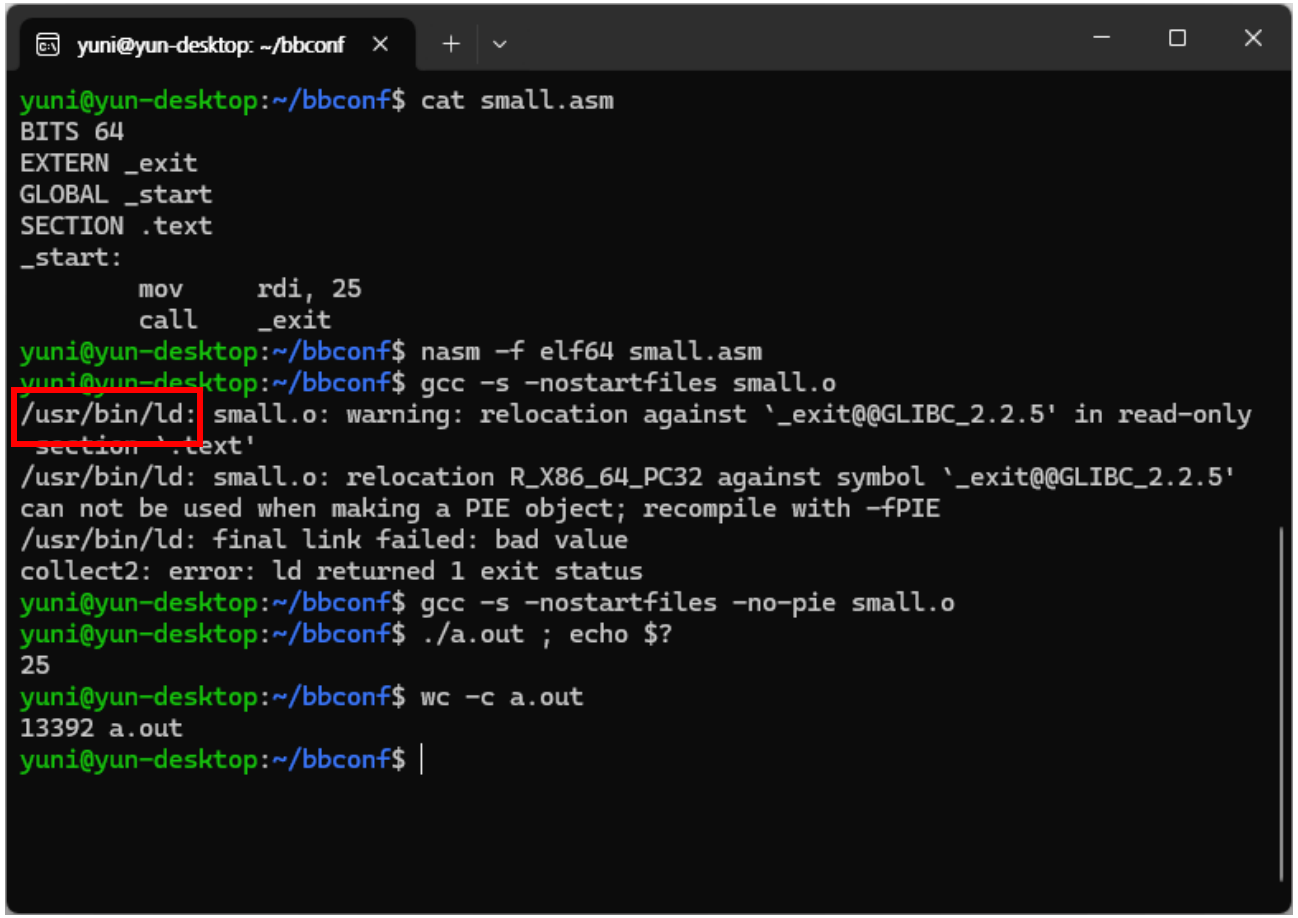
- 바이너리 파일 디스크립터 라이브러리 (libbfd)

각주 [편집]

1. ↑ GNU Manuals Online: binutils
2. ↑ Answer to the question "What do CC and LD stand for?"

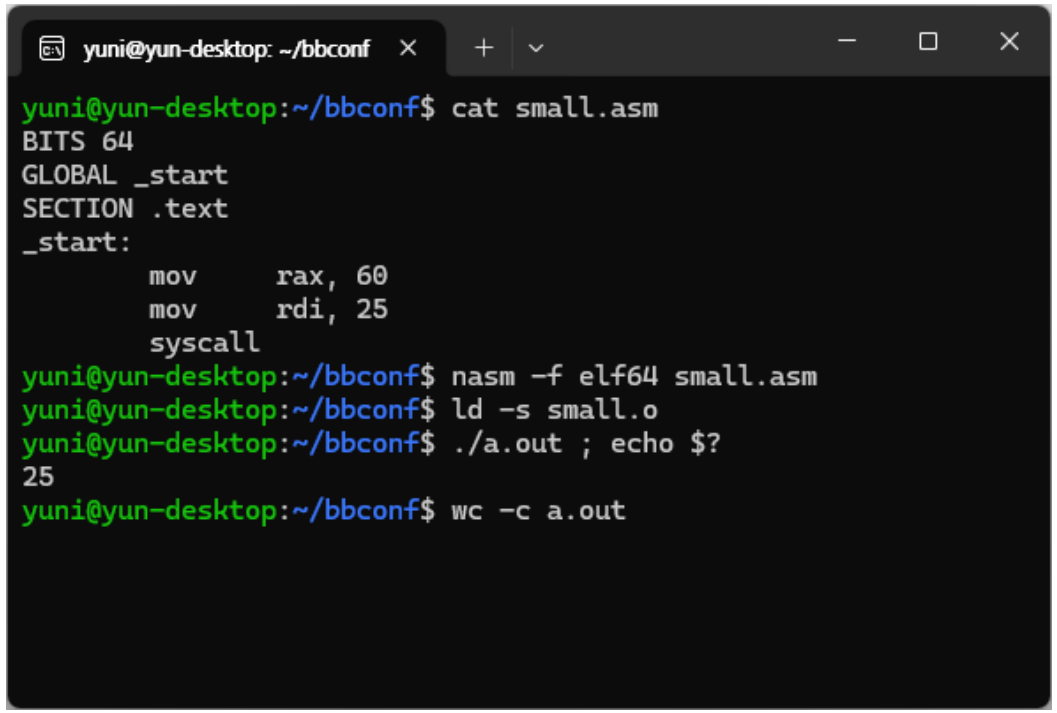
Best record **4400** Bytes

gcc는 버리고 ld로 가자



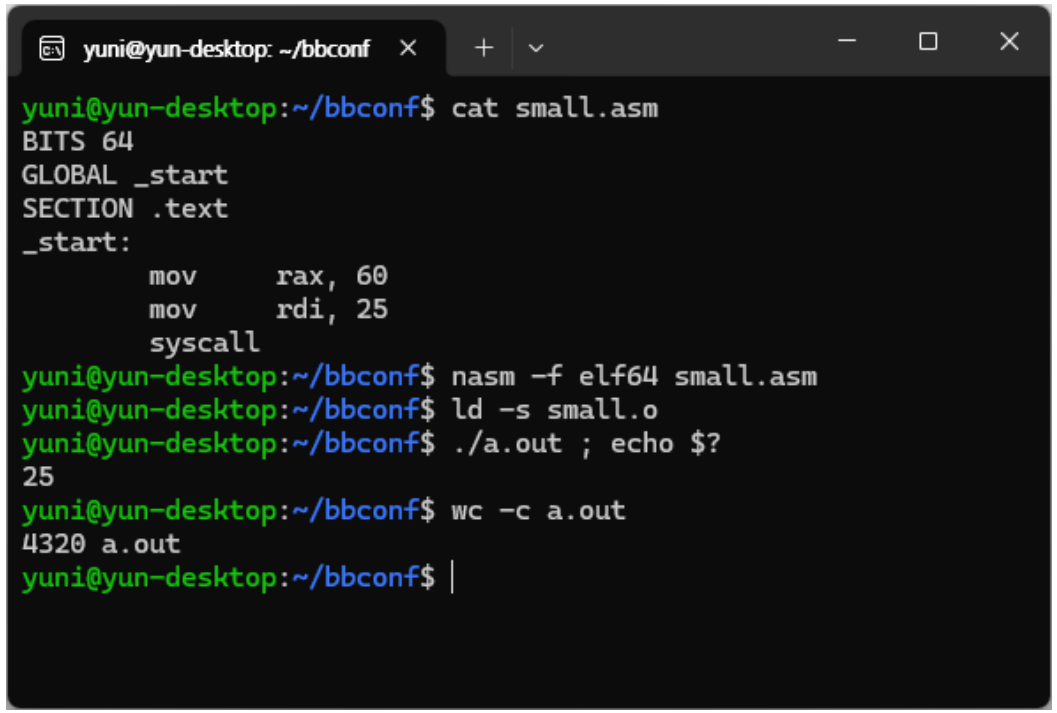
Best record **4400** Bytes

gcc는 버리고 ld로 가자



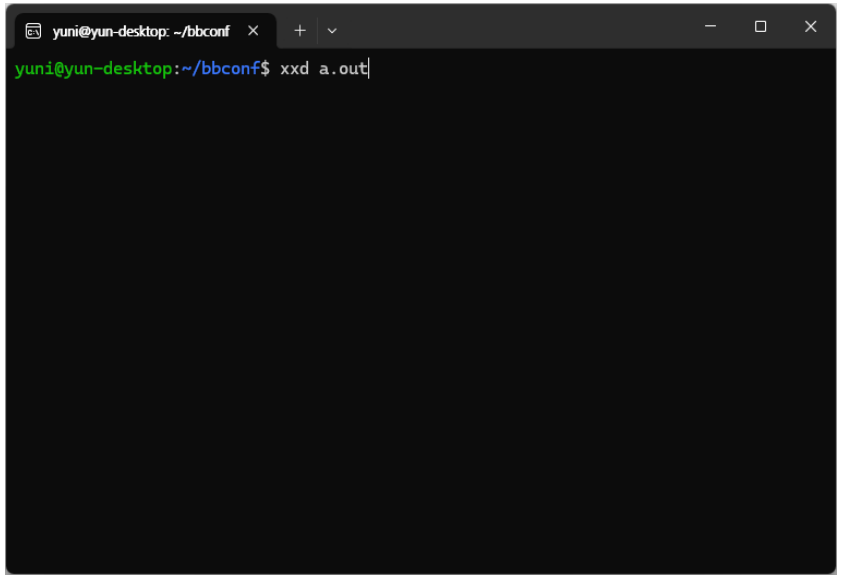
Best record **4400** Bytes

gcc는 버리고 ld로 가자



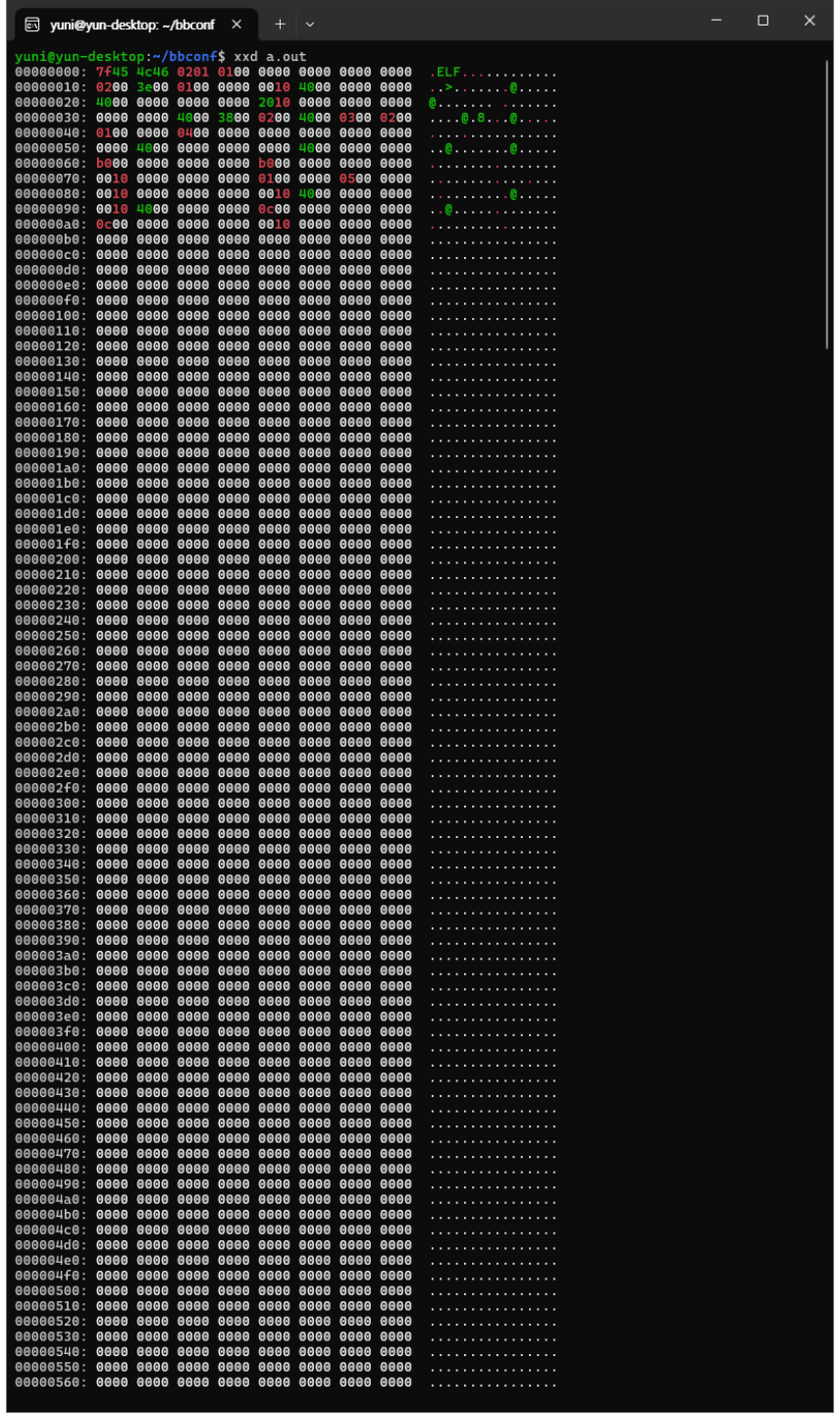
Best record **4400** -> **4320** Bytes (▼80)

실행파일 들여다 보기



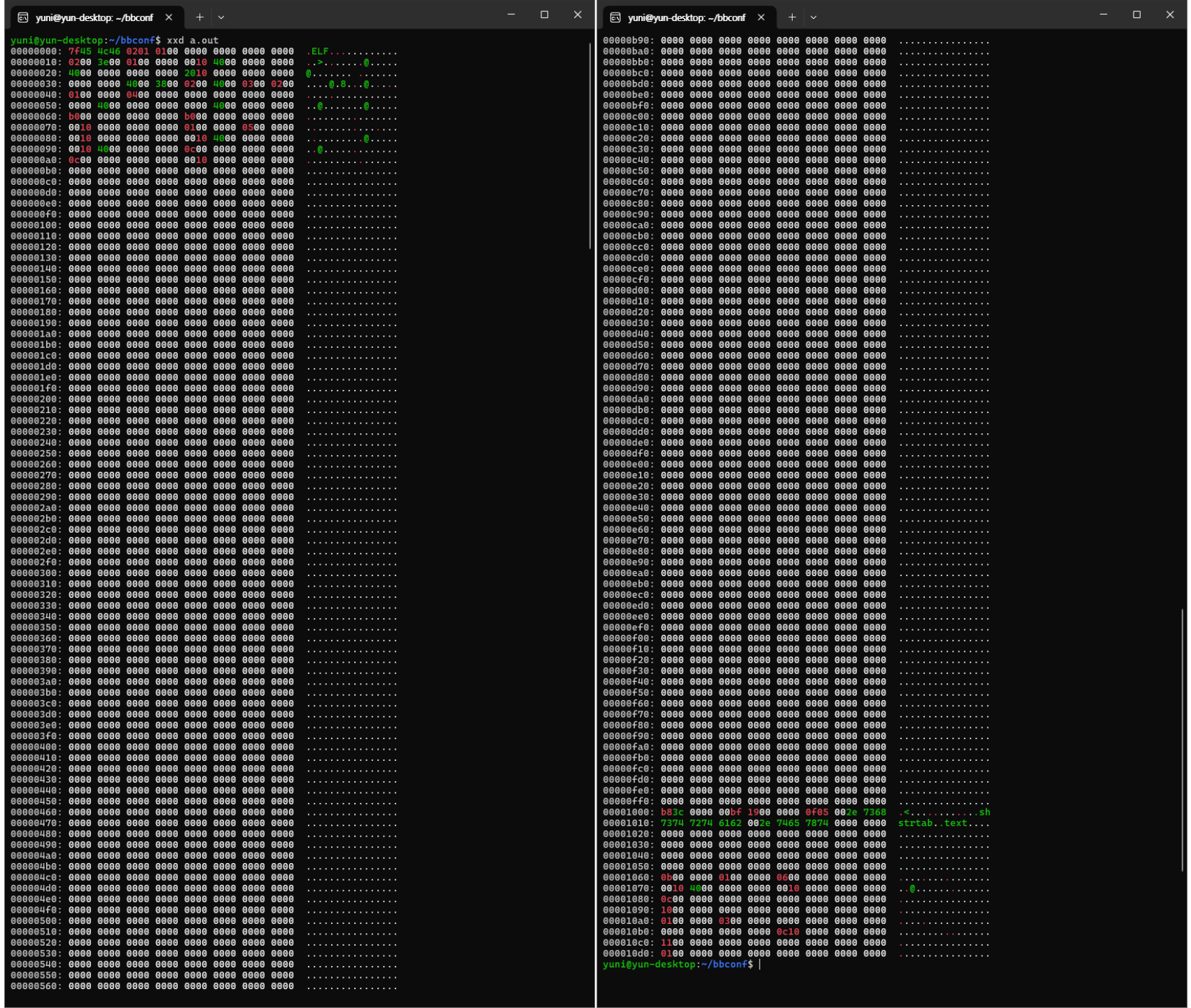
Best record **4320** Bytes

실행파일 들여다 보기



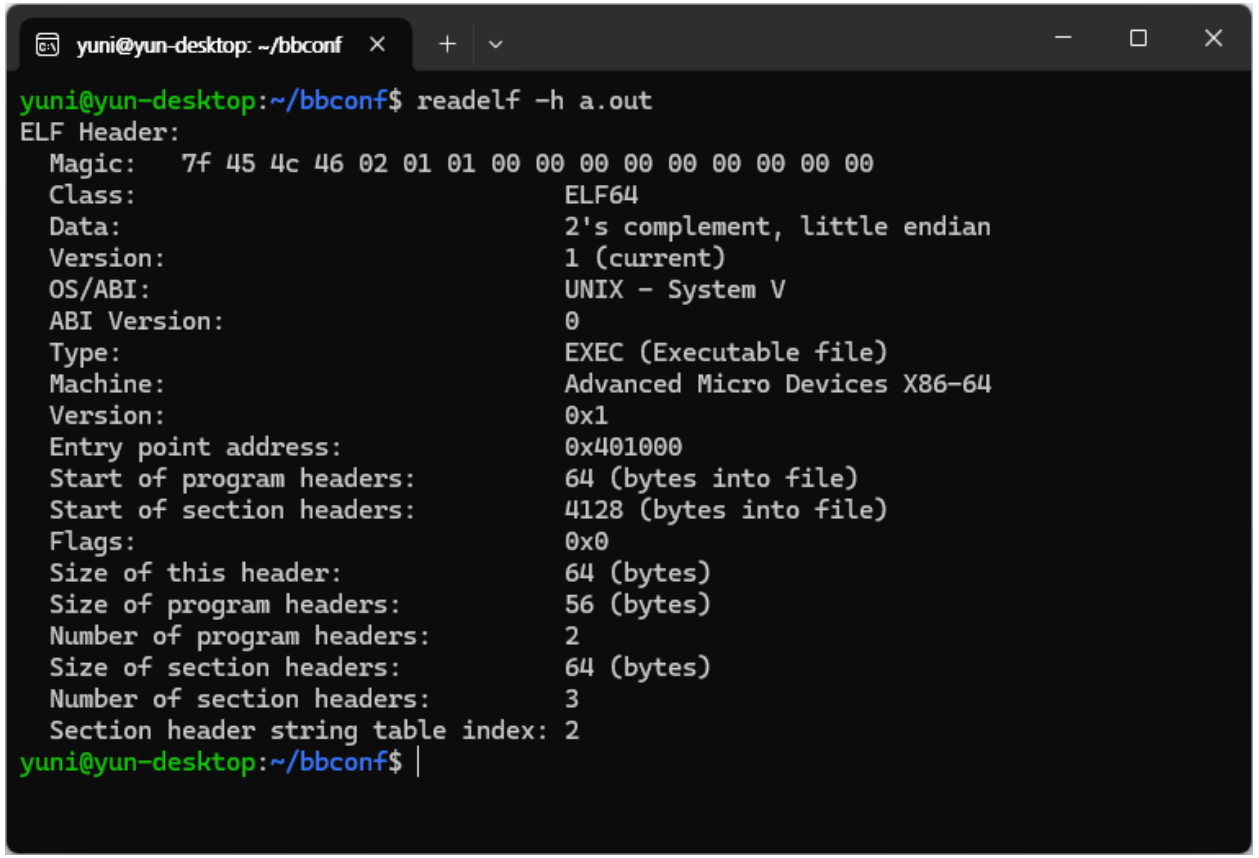
Best record **4320** Bytes

실행파일 들여다 보기



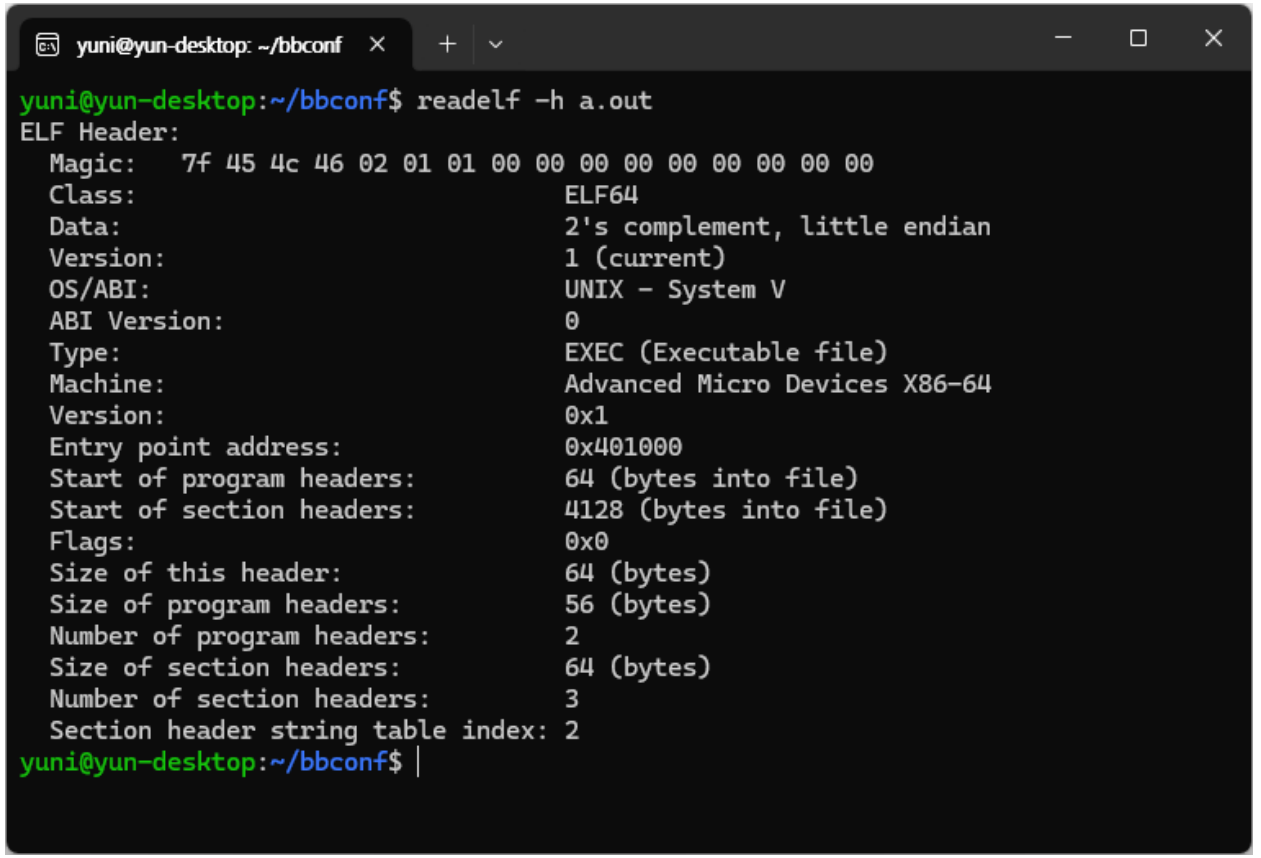
Best record **4320** Bytes

실행파일 들여다 보기



Best record **4320** Bytes

실행파일 들여다 보기



Section header 시작 지점이 왜 저런지 확인이 필요함

Best record **4320** Bytes

ELF 포맷을 까보자

3. File header

The file header is located at the beginning of the file, and is used to locate the other parts of the file. The structure is shown in Figure 2.

```
typedef struct
{
        unsigned char   e_ident[16];    /* ELF identification */
        Elf64_Half      e_type;         /* Object file type */
        Elf64_Half      e_machine;      /* Machine type */
        Elf64_Word      e_version;      /* Object file version */
        Elf64_Addr      e_entry;        /* Entry point address */
        Elf64_Off       e_phoff;        /* Program header offset */
        Elf64_Off       e_shoff;        /* Section header offset */
        Elf64_Word      e_flags;        /* Processor-specific flags */
        Elf64_Half      e_ehsize;       /* ELF header size */
        Elf64_Half      e_phentsize;    /* Size of program header entry */
        Elf64_Half      e_phnum;        /* Number of program header entries */
        Elf64_Half      e_shentsize;    /* Size of section header entry */
        Elf64_Half      e_shnum;        /* Number of section header entries */
        Elf64_Half      e_shstrndx;     /* Section name string table index */
} Elf64_Ehdr;
```

Figure 2. ELF-64 Header

Best record **4320** Bytes

ELF 포맷을 까보자

3. File header

The file header is located at the beginning of the file, and is used to locate the other parts of the file. The structure is shown in Figure 2.

```
typedef struct
{
        unsigned char   e_ident[16];    /* ELF identification */
        Elf64_Half      e_type;         /* Object file type */
        Elf64_Half      e_machine;      /* Machine type */
        Elf64_Word      e_version;      /* Object file version */
        Elf64_Addr      e_entry;        /* Entry point address */
        Elf64_Off       e_phoff;        /* Program header offset */
        Elf64_Off       e_shoff;        /* Section header offset */
        Elf64_Word      e_flags;        /* Processor-specific flags */
        Elf64_Half      e_ehsize;       /* ELF header size */
        Elf64_Half      e_phentsize;    /* Size of program header entry */
        Elf64_Half      e_phnum;        /* Number of program header entries */
        Elf64_Half      e_shentsize;    /* Size of section header entry */
        Elf64_Half      e_shnum;        /* Number of section header entries */
        Elf64_Half      e_shstrndx;     /* Section name string table index */
} Elf64_Ehdr;
```

Figure 2. ELF-64 Header

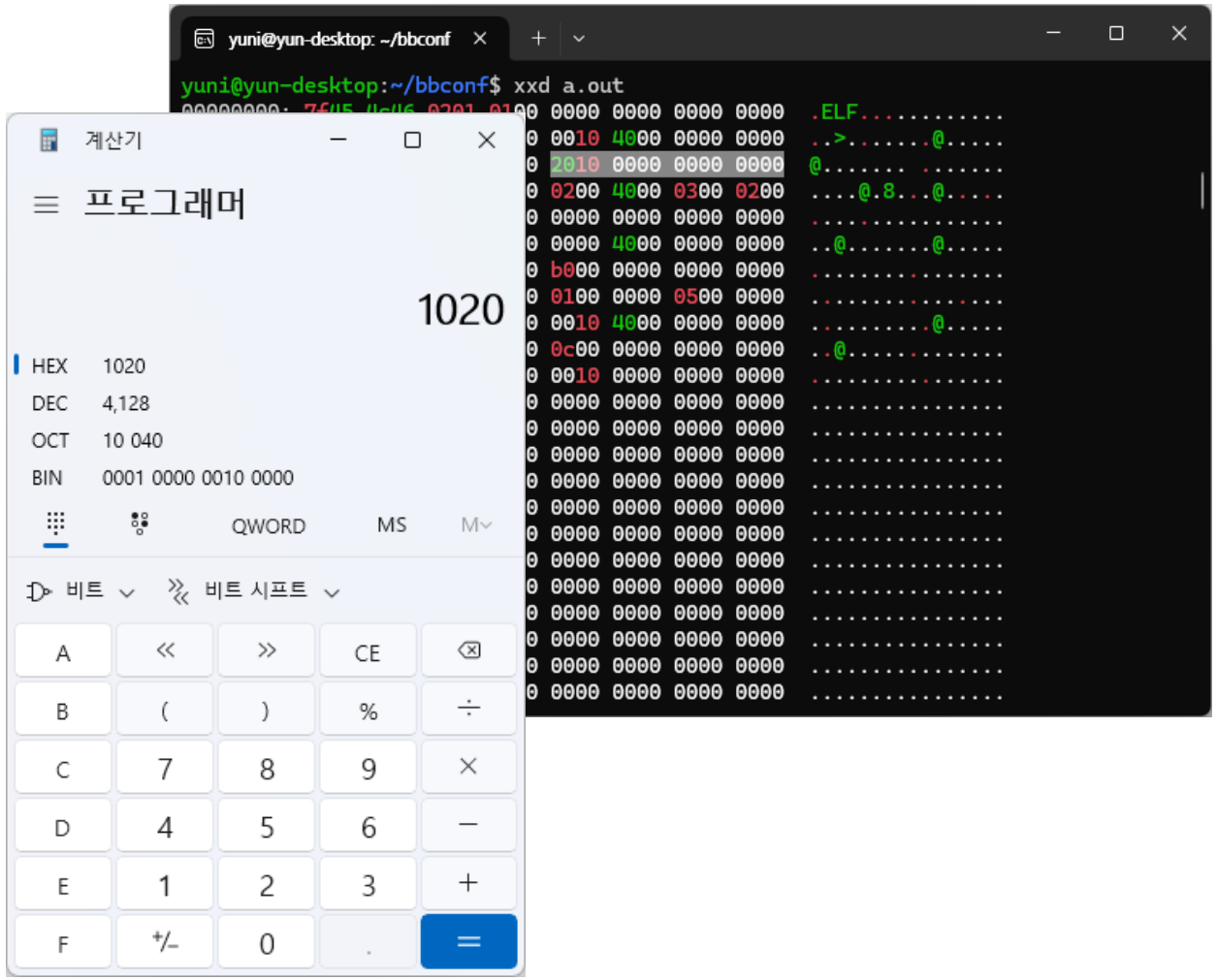
e_shoff는 0x28부터 시작

Best record **4320** Bytes

ELF 포맷을 까보자

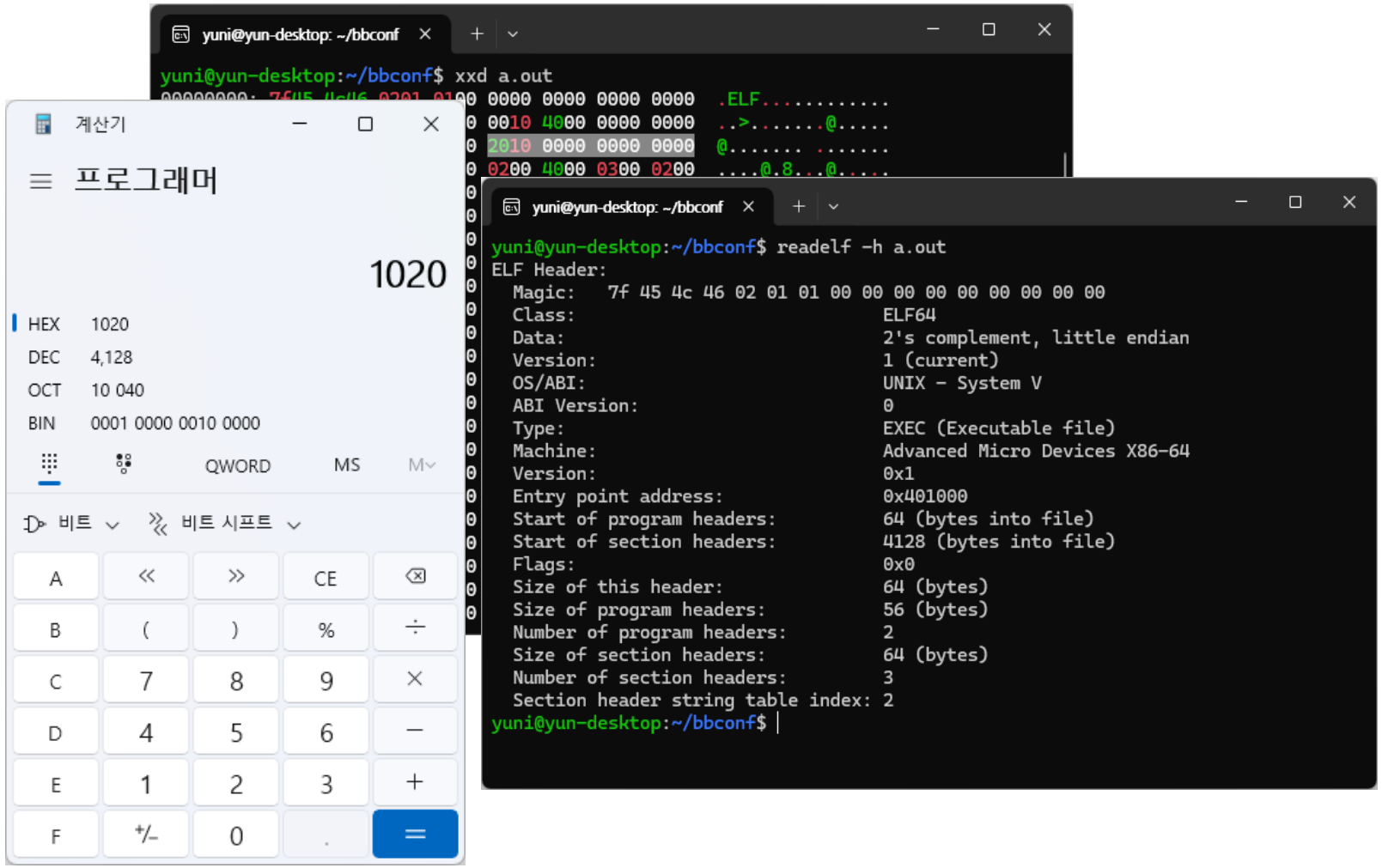
Best record **4320** Bytes

ELF 포맷을 까보자



Best record **4320** Bytes

ELF 포맷을 까보자



Best record **4320** Bytes

ELF 포맷을 까보자

```
yuni@yun-desktop:~/bbconf$ readelf -h a.out
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x401000
  Start of program headers:          64 (bytes into file)
  Start of section headers:          4128 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         2
  Size of section headers:           64 (bytes)
  Number of section headers:         3
  Section header string table index: 2
yuni@yun-desktop:~/bbconf$
```

```
yuni@yun-desktop:~/bbconf$ xxd a.out
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
00000010: 0200 3e00 0100 0000 0010 4000 0000 0000  ..>.......@.....
00000020: 4000 0000 0000 0000 2010 0000 0000 0000  @....... .......
00000030: 0000 0000 4000 3800 0200 4000 0300 0200  ....@.8...@.....
00000040: 0100 0000 0400 0000 0000 0000 0000 0000  ................
00000050: 0000 4000 0000 0000 0000 4000 0000 0000  ..@.......@.....
00000060: b000 0000 0000 0000 b000 0000 0000 0000  ................
00000070: 0010 0000 0000 0000 0100 0000 0500 0000  ................
00000080: 0010 0000 0000 0000 0010 4000 0000 0000  ..........@.....
00000090: 0010 4000 0000 0000 0c00 0000 0000 0000  ..@.............
000000a0: 0c00 0000 0000 0000 0010 0000 0000 0000  ................
000000b0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000c0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000d0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000e0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000f0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000100: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000110: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000120: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000130: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000140: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000150: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000160: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

ELF header

Program header

Best record **4320** Bytes

ELF 포맷을 까보자

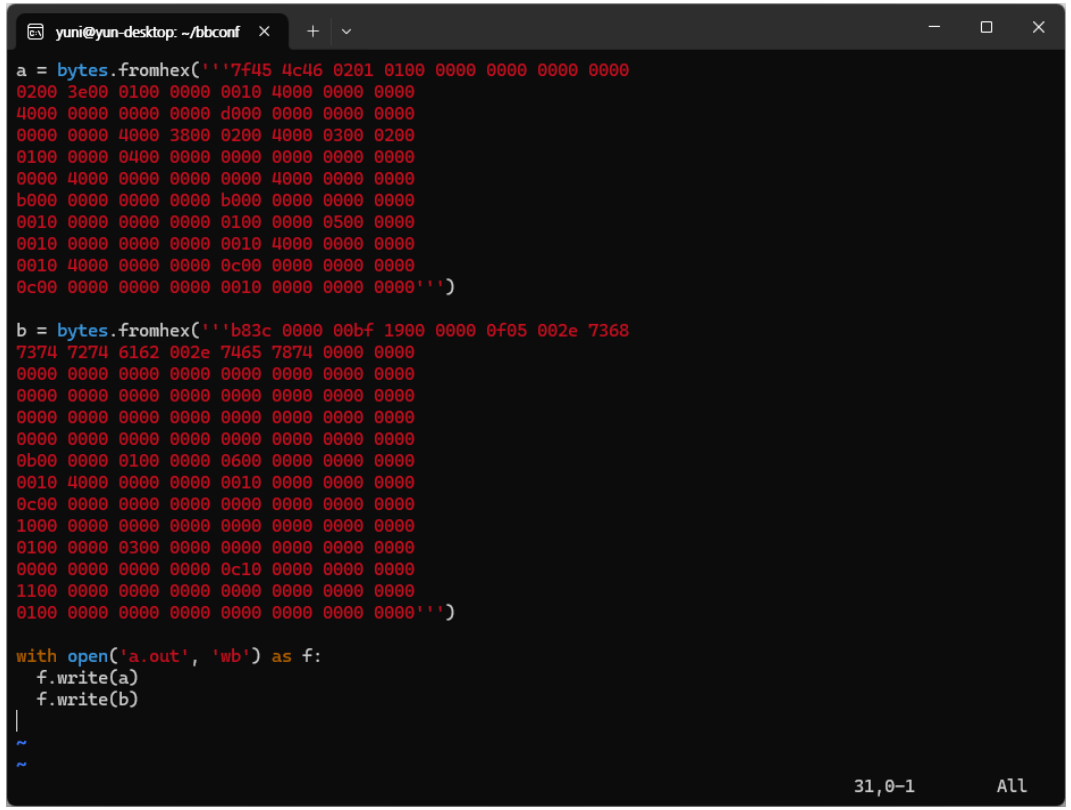
```
yuni@yun-desktop:~/bbconf$ readelf -h a.out
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x401000
  Start of program headers:          64 (bytes into file)
  Start of section headers:          4128 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         2
  Size of section headers:           64 (bytes)
  Number of section headers:         3
  Section header string table index: 2
yuni@yun-desktop:~/bbconf$
```

```
00000f00: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000f10: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000f20: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000f30: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000f40: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000f50: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000f60: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000f70: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000f80: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000f90: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000fa0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000fb0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000fc0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000fd0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000fe0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000ff0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00001000: b83c 0000 00bf 1900 0000 0f05 002e 7368  .<............sh
00001010: 7374 7274 6162 002e 7465 7874 0000 0000  strtab..text....
00001020: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00001030: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00001040: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00001050: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00001060: 0b00 0000 0100 0000 0600 0000 0000 0000  ................
00001070: 0010 4000 0000 0000 0010 0000 0000 0000  ..@.............
00001080: 0c00 0000 0000 0000 0000 0000 0000 0000  ................
00001090: 1000 0000 0000 0000 0000 0000 0000 0000  ................
000010a0: 0100 0000 0300 0000 0000 0000 0000 0000  ................
000010b0: 0000 0000 0000 0000 0c10 0000 0000 0000  ................
000010c0: 1100 0000 0000 0000 0000 0000 0000 0000  ................
000010d0: 0100 0000 0000 0000 0000 0000 0000 0000  ................
yuni@yun-desktop:~/bbconf$
```

Section header

Best record **4320** Bytes

ELF 포맷을 까보자

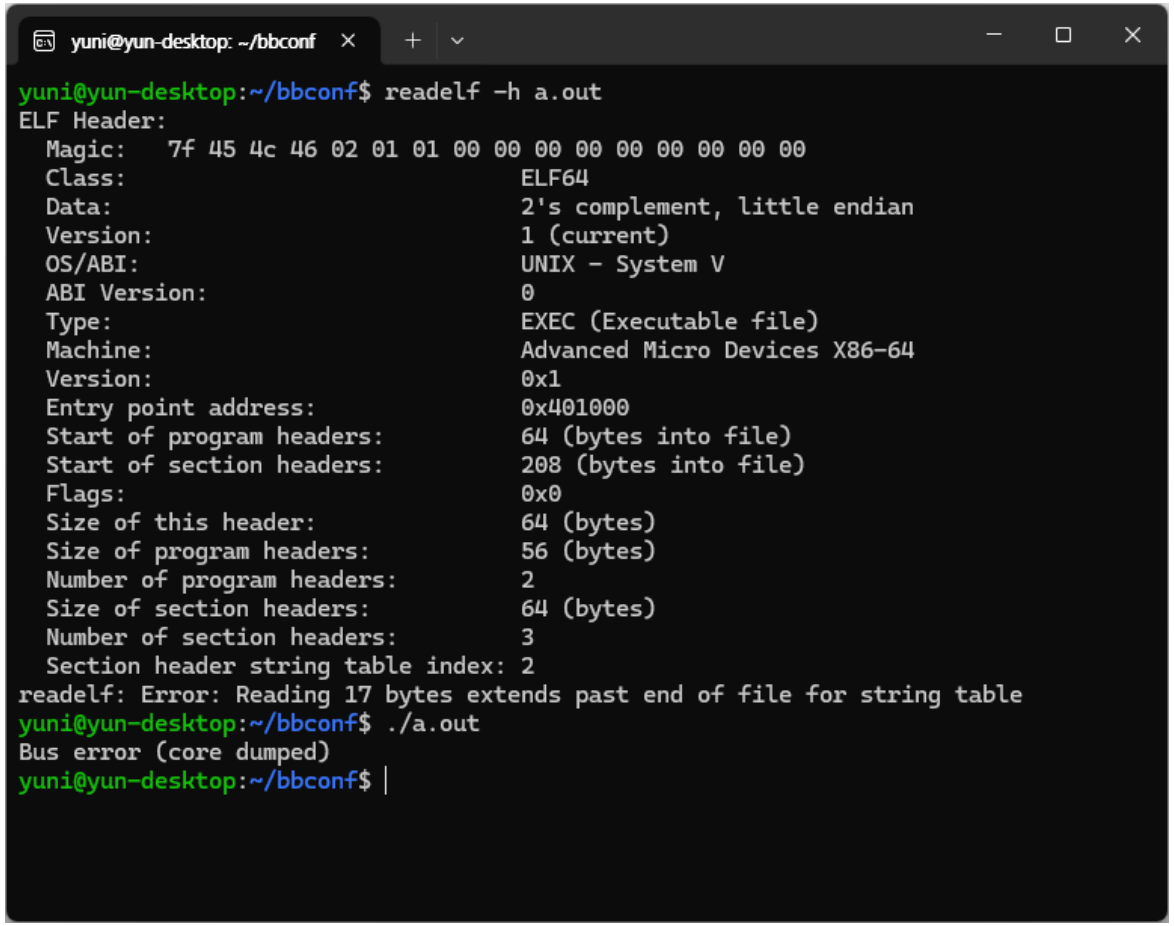


Best record **4320** Bytes

ELF 포맷을 까보자

Best record **4320** Bytes

ELF 포맷을 까보자



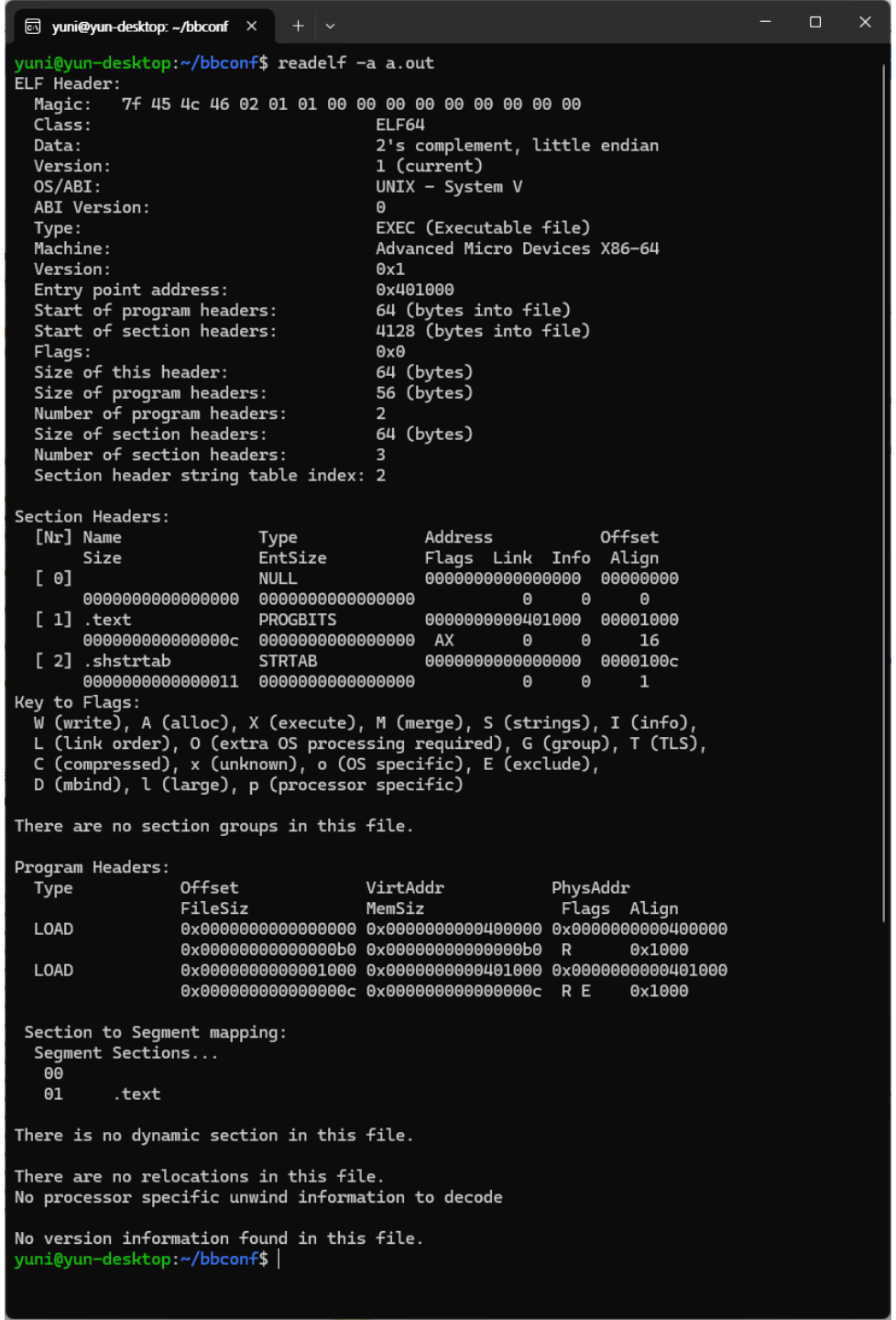
Best record **4320** Bytes

ELF 포맷을 까보자


```
yuni@yun-desktop: ~/bbconf
yuni@yun-desktop:~/bbconf$ readelf -h a.out
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x401000
  Start of program headers:          64 (bytes into file)
  Start of section headers:          208 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         2
  Size of section headers:           64 (bytes)
  Number of section headers:         3
  Section header string table index: 2
readelf: Error: Reading 17 bytes extends past end of file for string table
yuni@yun-desktop:~/bbconf$ ./a.out
Bus error (core dumped)
yuni@yun-desktop:~/bbconf$
```

Best record **4320** Bytes

ELF 포맷을 까보자



Best record **4320** Bytes

ELF 포맷을 까보자

Figure 5-1: Program Header

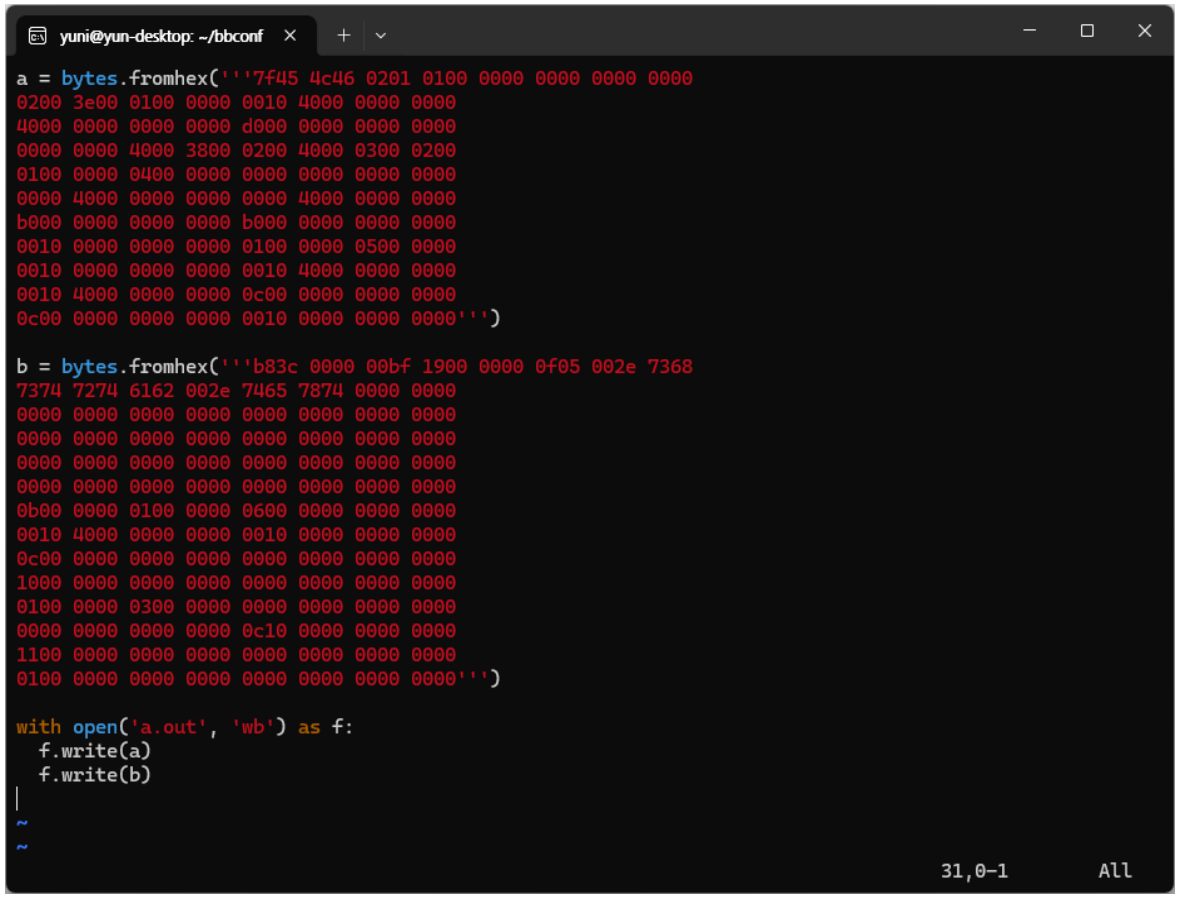
```
typedef struct {
        Elf32_Word      p_type;
        Elf32_Off       p_offset;
        Elf32_Addr      p_vaddr;
        Elf32_Addr      p_paddr;
        Elf32_Word      p_filesz;
        Elf32_Word      p_memsz;
        Elf32_Word      p_flags;
        Elf32_Word      p_align;
} Elf32_Phdr;

typedef struct {
        Elf64_Word      p_type;
        Elf64_Word      p_flags;
        Elf64_Off       p_offset;
        Elf64_Addr      p_vaddr;
        Elf64_Addr      p_paddr;
        Elf64_Xword     p_filesz;
        Elf64_Xword     p_memsz;
        Elf64_Xword     p_align;
} Elf64_Phdr;
```

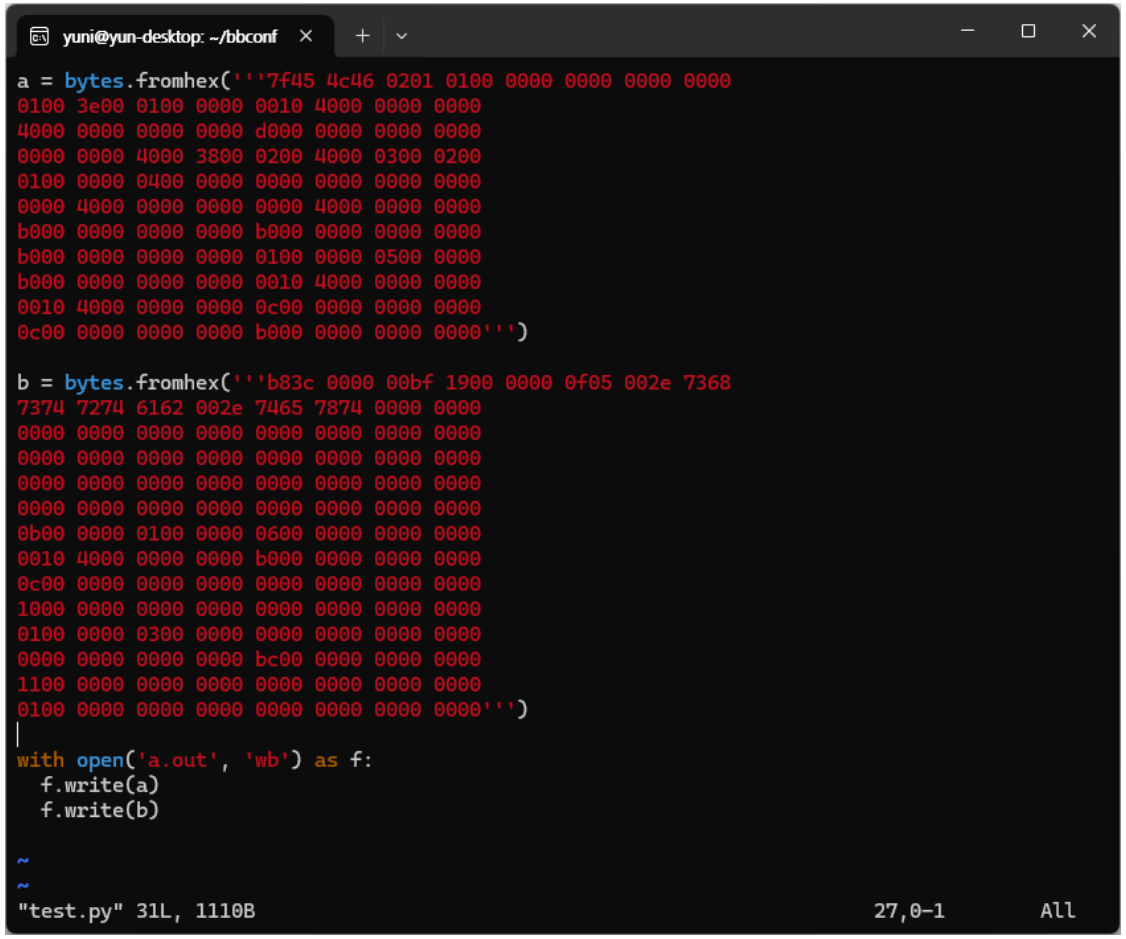
Figure 4-8: Section Header

```
typedef struct {
        Elf32_Word      sh_name;
        Elf32_Word      sh_type;
        Elf32_Word      sh_flags;
        Elf32_Addr      sh_addr;
        Elf32_Off       sh_offset;
        Elf32_Word      sh_size;
        Elf32_Word      sh_link;
        Elf32_Word      sh_info;
        Elf32_Word      sh_addralign;
        Elf32_Word      sh_entsize;
} Elf32_Shdr;

typedef struct {
        Elf64_Word      sh_name;
        Elf64_Word      sh_type;
        Elf64_Xword     sh_flags;
        Elf64_Addr      sh_addr;
        Elf64_Off       sh_offset;
        Elf64_Xword     sh_size;
        Elf64_Word      sh_link;
        Elf64_Word      sh_info;
        Elf64_Xword     sh_addralign;
        Elf64_Xword     sh_entsize;
} Elf64_Shdr;
```

Best record **4320** Bytes

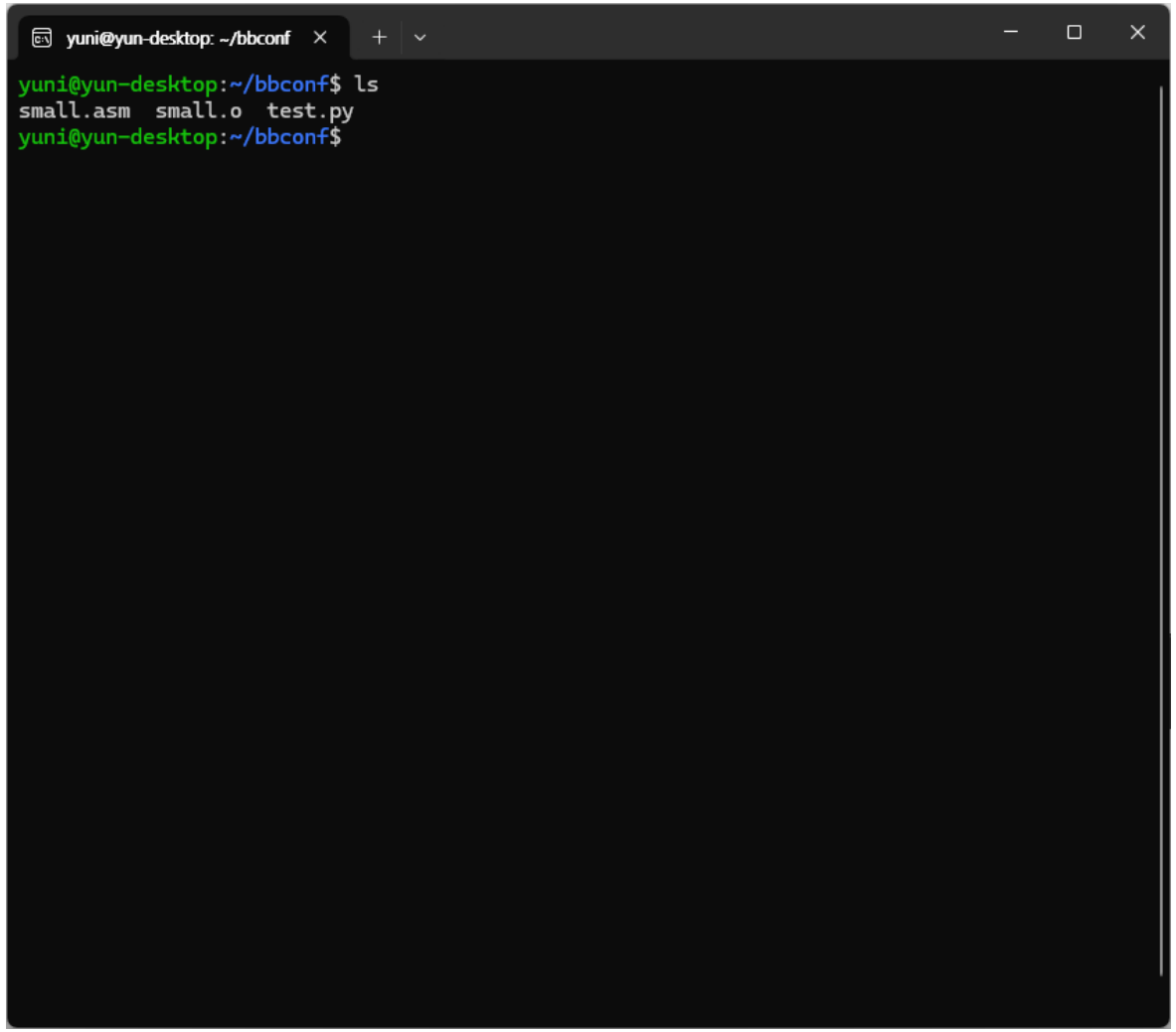
ELF 포맷을 까보자





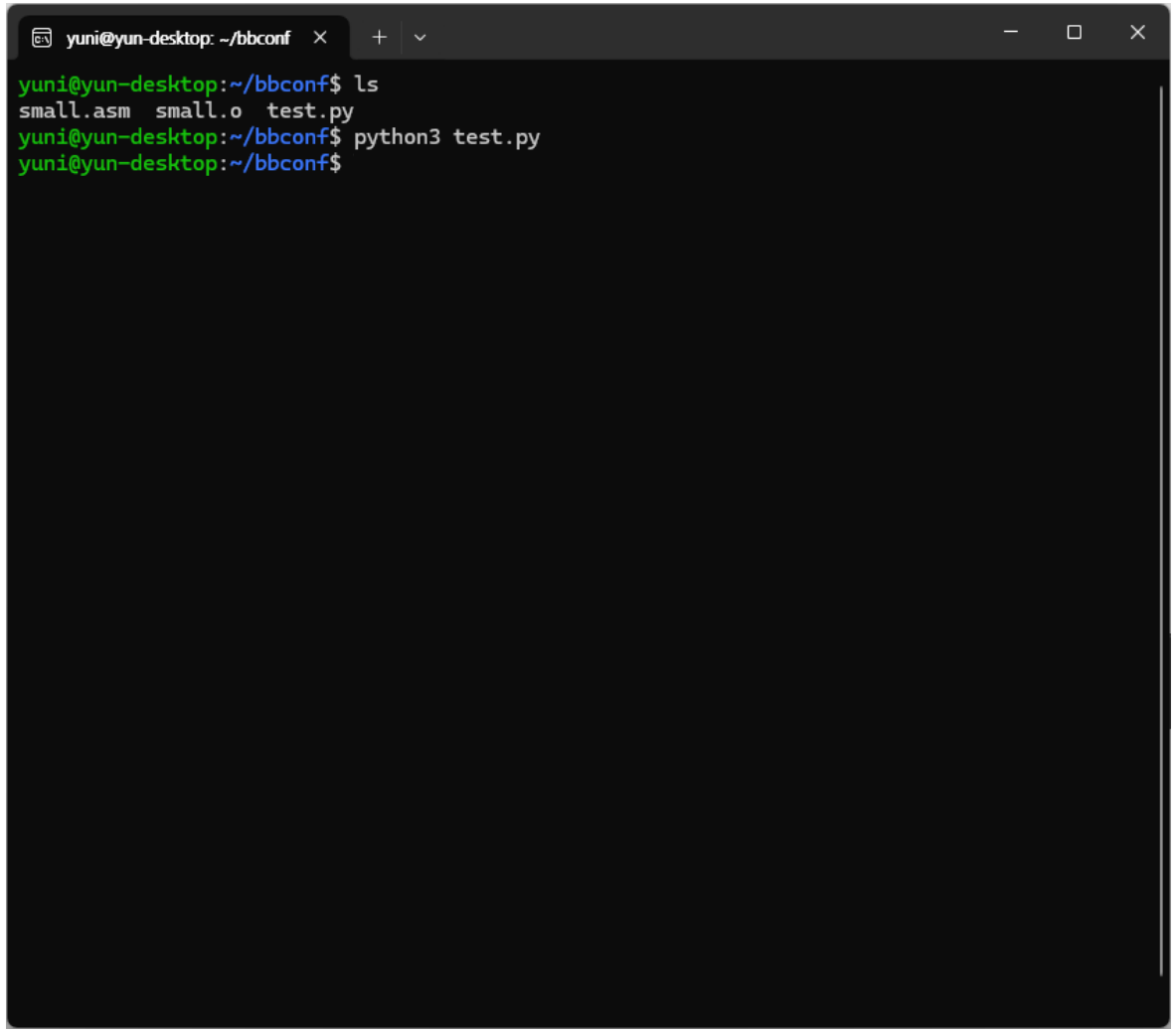
Best record **4320** Bytes

ELF 포맷을 까보자



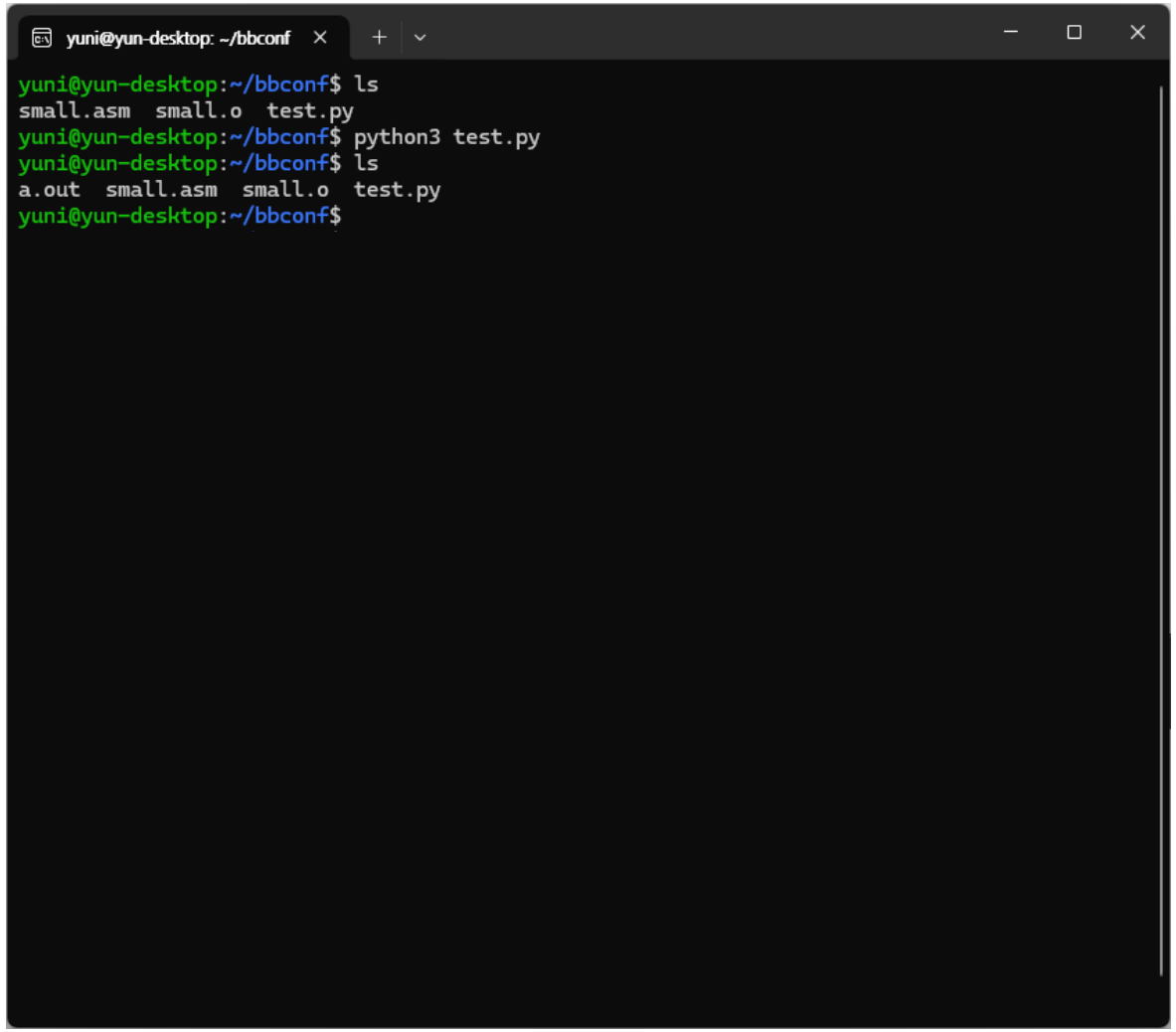
Best record **4320** Bytes

ELF 포맷을 까보자



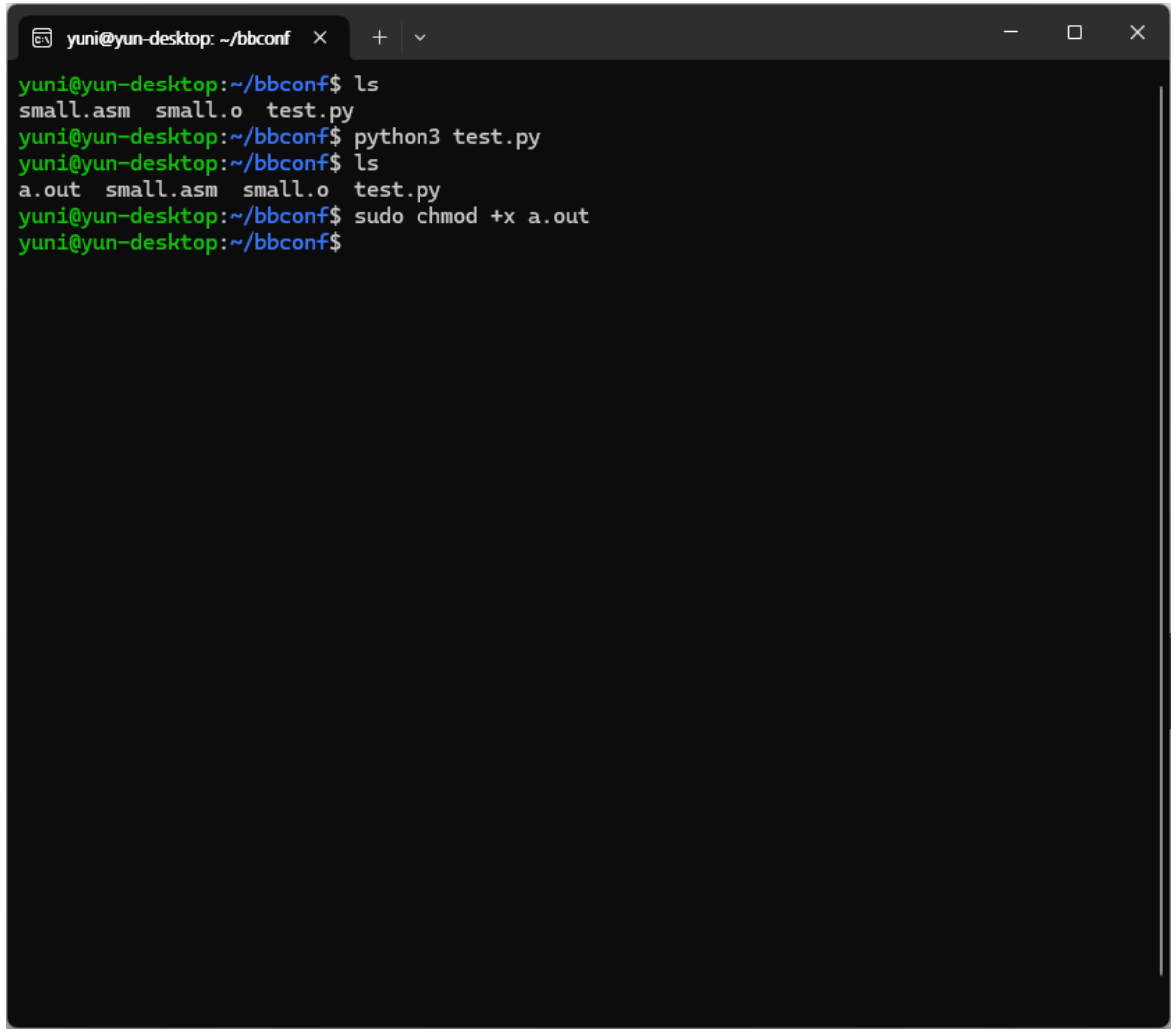
Best record **4320** Bytes

ELF 포맷을 까보자



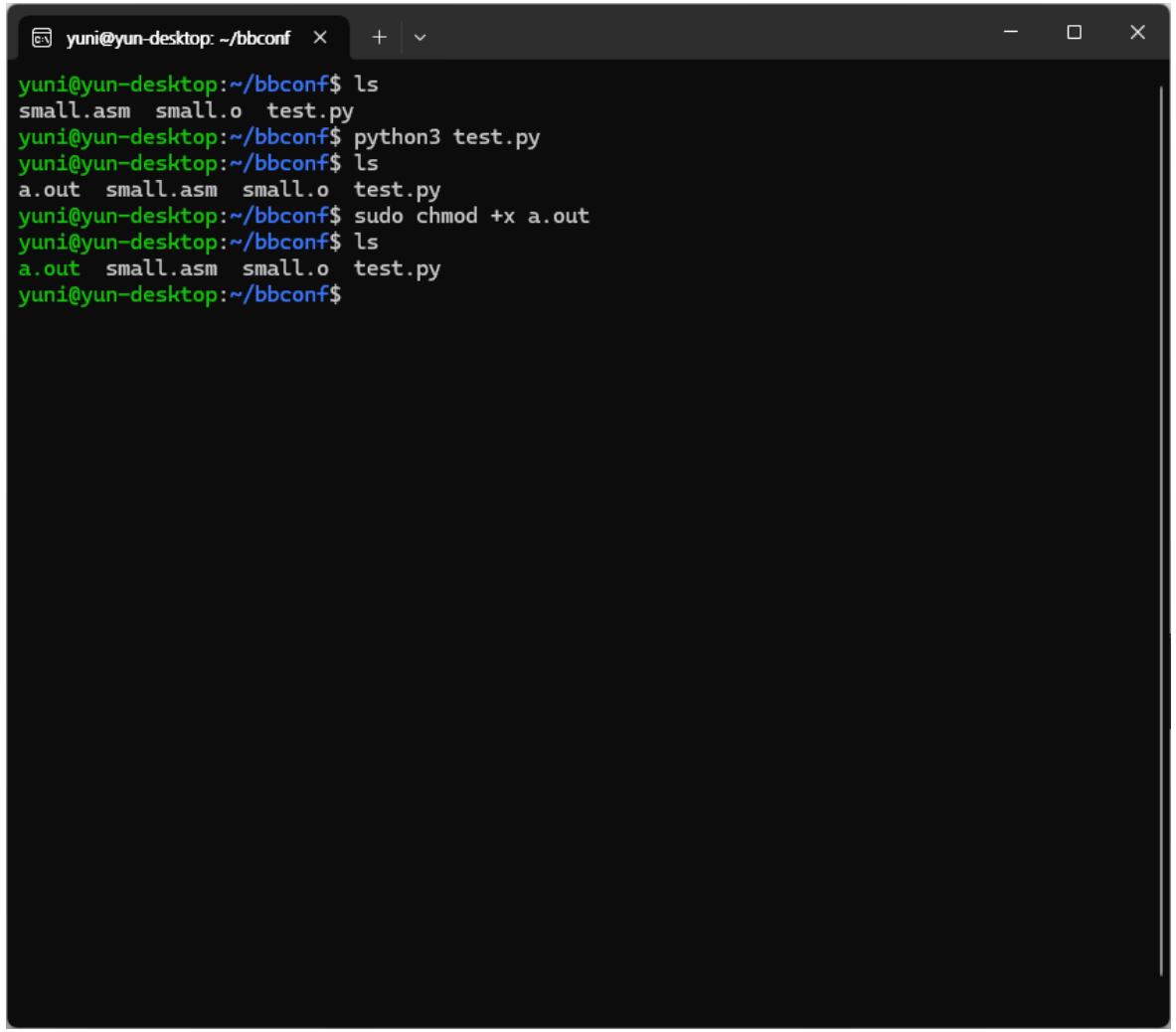
Best record **4320** Bytes

ELF 포맷을 까보자



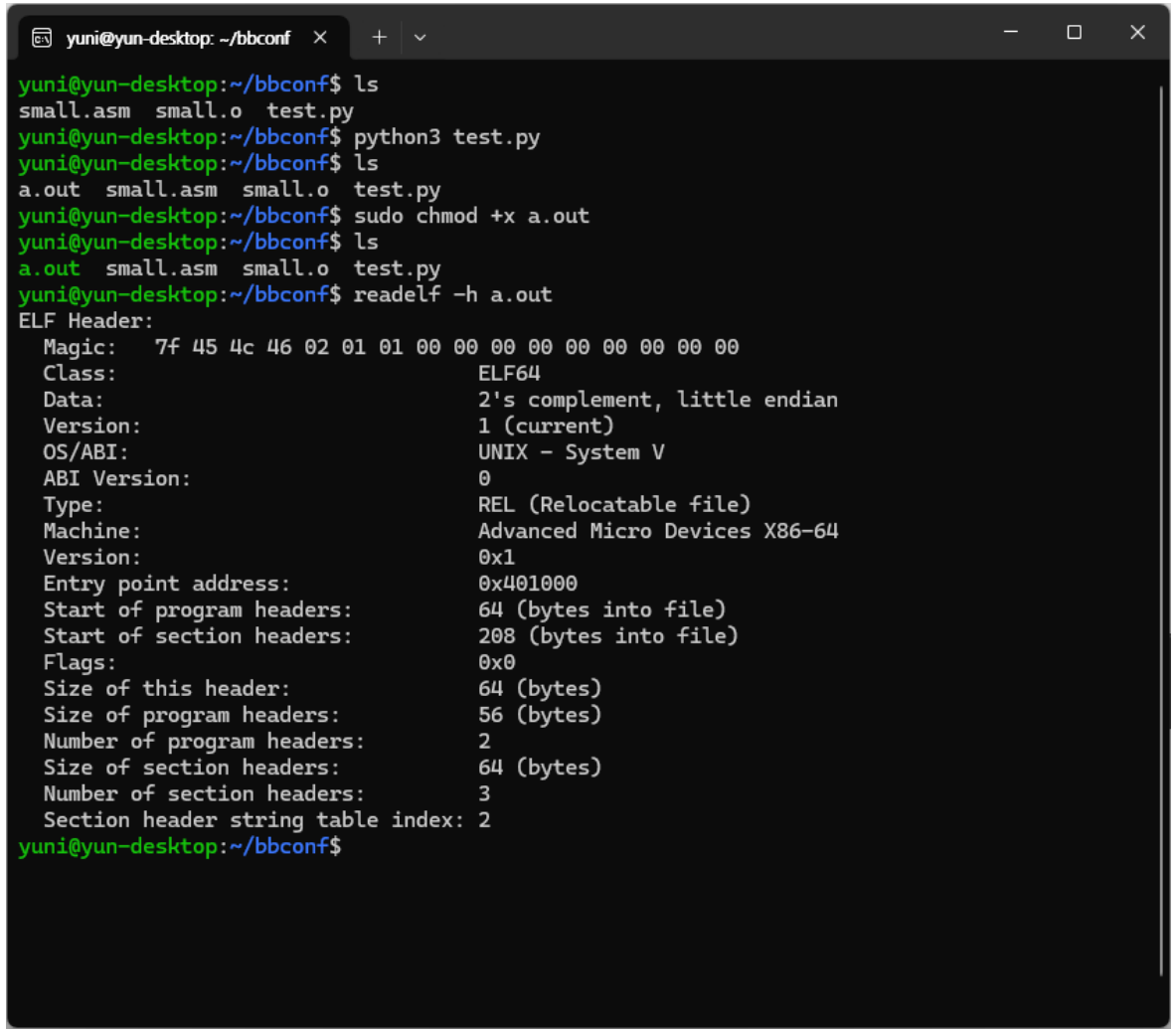
Best record **4320** Bytes

ELF 포맷을 까보자



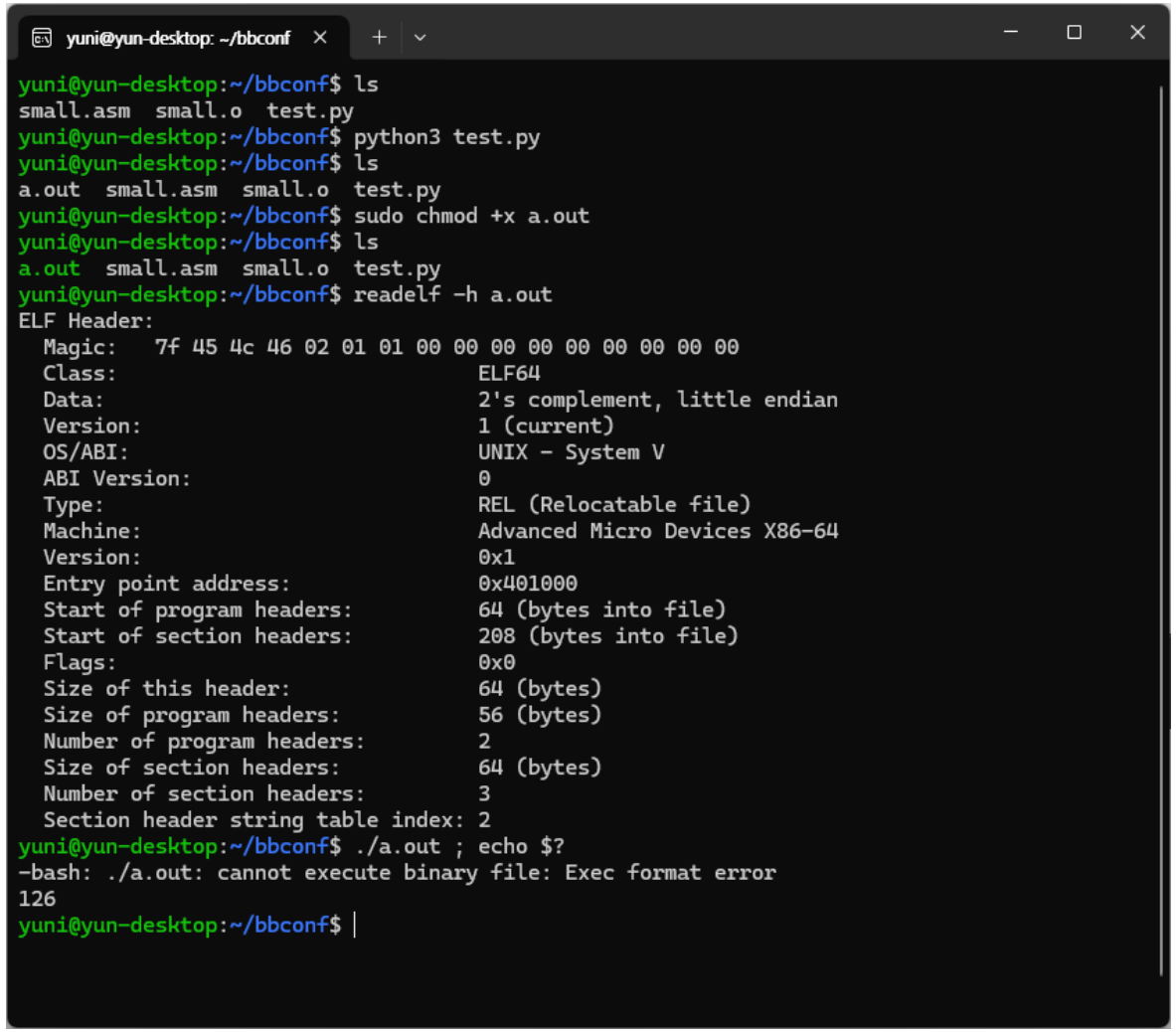
Best record **4320** Bytes

ELF 포맷을 까보자



Best record **4320** Bytes

ELF 포맷을 까보자



Best record **4320** Bytes

Header의 필요성

Program Header

Section Header

Best record **4320** Bytes

Header의 필요성

Program Header

주로 ELF 헤더 바로 뒤에 위치

Section Header

주로 file의 끝이나 끝 근처에 위치

Header의 필요성

Program Header

주로 ELF 헤더 바로 뒤에 위치

프로그램의 각 부분이 메모리 어디에 로드되어야 하는지를 서술

Section Header

주로 file의 끝이나 끝 근처에 위치

프로그램의 각 부분이 파일 내에서 어디에 있는지 식별하는 데 사용

Header의 필요성

Program Header

주로 ELF 헤더 바로 뒤에 위치

프로그램의 각 부분이 메모리 어디에 로드되어야 하는지를 서술

로더가 사용함

Section Header

주로 file의 끝이나 끝 근처에 위치

프로그램의 각 부분이 파일 내에서 어디에 있는지 식별하는 데 사용

컴파일러, 링커가 사용함

Header의 필요성

Program Header

주로 ELF 헤더 바로 뒤에 위치

프로그램의 각 부분이 메모리 어디에 로드되어야 하는지를 서술

로더가 사용함

Optional for object files

Section Header

주로 file의 끝이나 끝 근처에 위치

프로그램의 각 부분이 파일 내에서 어디에 있는지 식별하는 데 사용

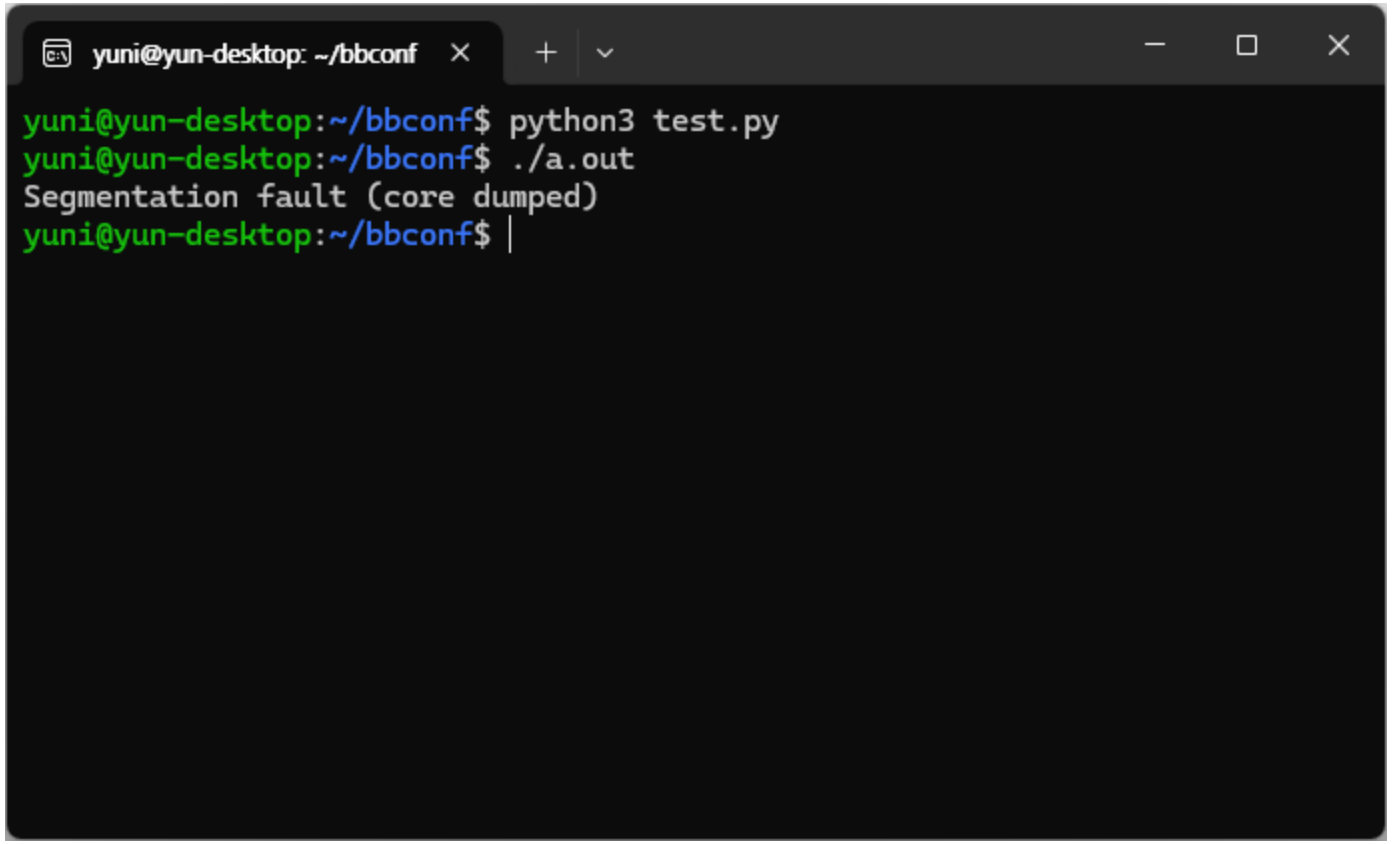
컴파일러, 링커가 사용함

Optional for executables

Header의 필요성

Best record **4320** Bytes

Header의 필요성



Best record **4320** Bytes

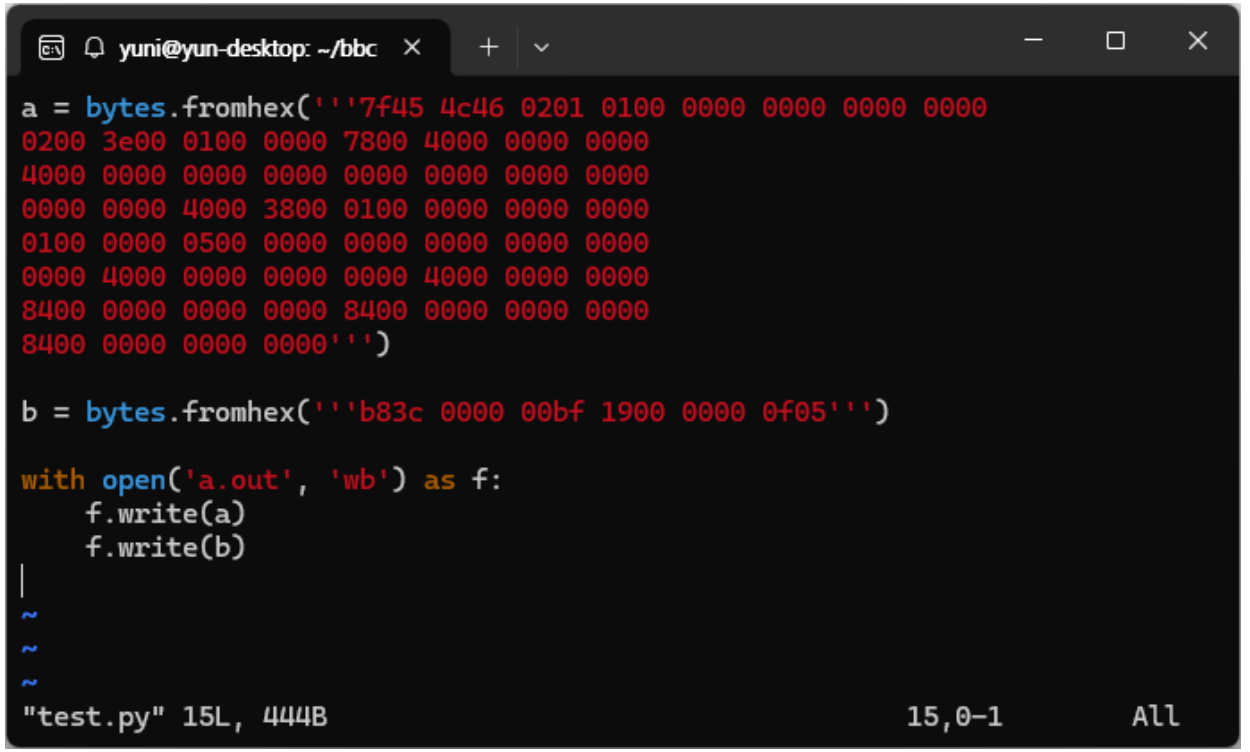
Header의 필요성

Program header[10]

Offset		Size (bytes)		Field	Purpose
32-bit	64-bit	32-bit	64-bit		
0x00		4		p_type	Identifies the type of the segment. Value / Name / Meaning 0x00000000 / PT_NULL / Program header table entry unused. 0x00000001 / PT_LOAD / Loadable segment. 0x00000002 / PT_DYNAMIC / Dynamic linking information. 0x00000003 / PT_INTERP / Interpreter information. 0x00000004 / PT_NOTE / Auxiliary information. 0x00000005 / PT_SHLIB / Reserved. 0x00000006 / PT_PHDR / Segment containing program header table itself. 0x00000007 / PT_TLS / Thread-Local Storage template. 0x60000000 / PT_LOOS / Reserved inclusive range. Operating system specific. 0x6FFFFFFF / PT_HIOS 0x70000000 / PT_LOPROC / Reserved inclusive range. Processor specific. 0x7FFFFFFF / PT_HIPROC
	0x04		4	p_flags	Segment-dependent flags (position for 64-bit structure). Value / Name / Meaning 0x1 / PF_X / Executable segment. 0x2 / PF_W / Writeable segment. 0x4 / PF_R / Readable segment.
0x04	0x08	4	8	p_offset	Offset of the segment in the file image.
0x08	0x10	4	8	p_vaddr	Virtual address of the segment in memory.
0x0C	0x18	4	8	p_paddr	On systems where physical address is relevant, reserved for segment's physical address.
0x10	0x20	4	8	p_filesz	Size in bytes of the segment in the file image. May be 0.
0x14	0x28	4	8	p_memsz	Size in bytes of the segment in memory. May be 0.
0x18		4		p_flags	Segment-dependent flags (position for 32-bit structure). See above `p_flags` field for flag definitions.
0x1C	0x30	4	8	p_align	`0` and `1` specify no alignment. Otherwise should be a positive, integral power of 2, with p_vaddr equating p_offset modulus p_align.
0x20	0x38				End of Program Header (size).

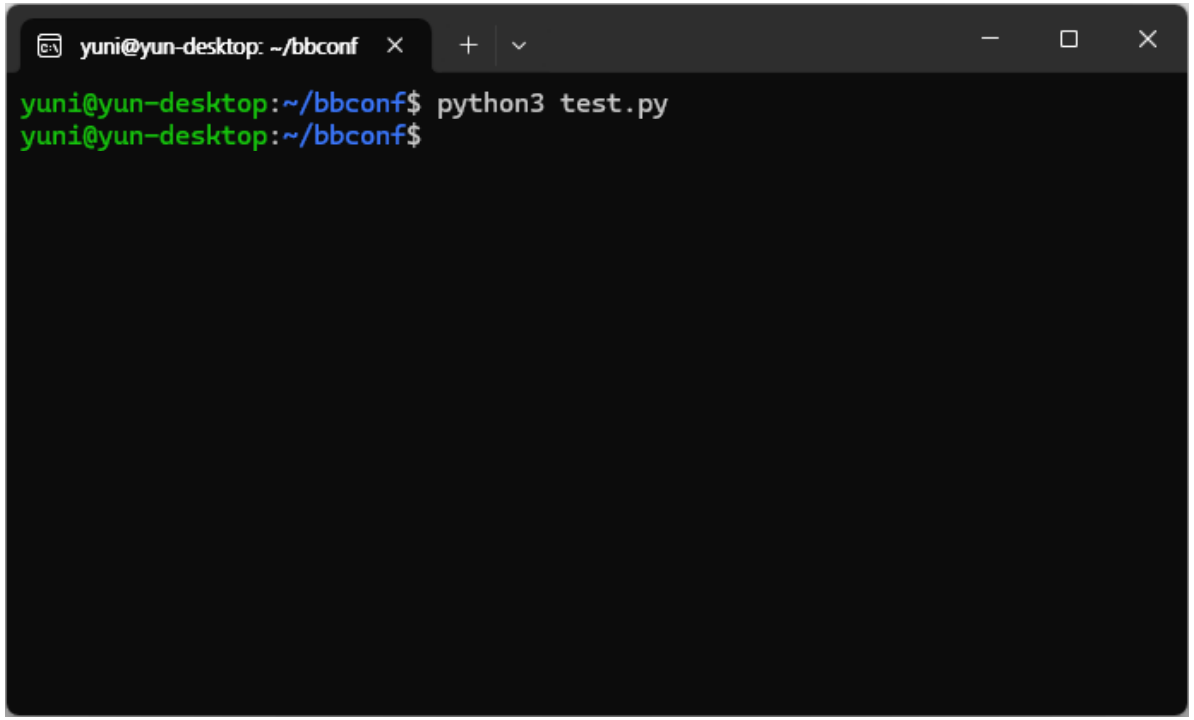
Best record **4320** Bytes

Header의 필요성



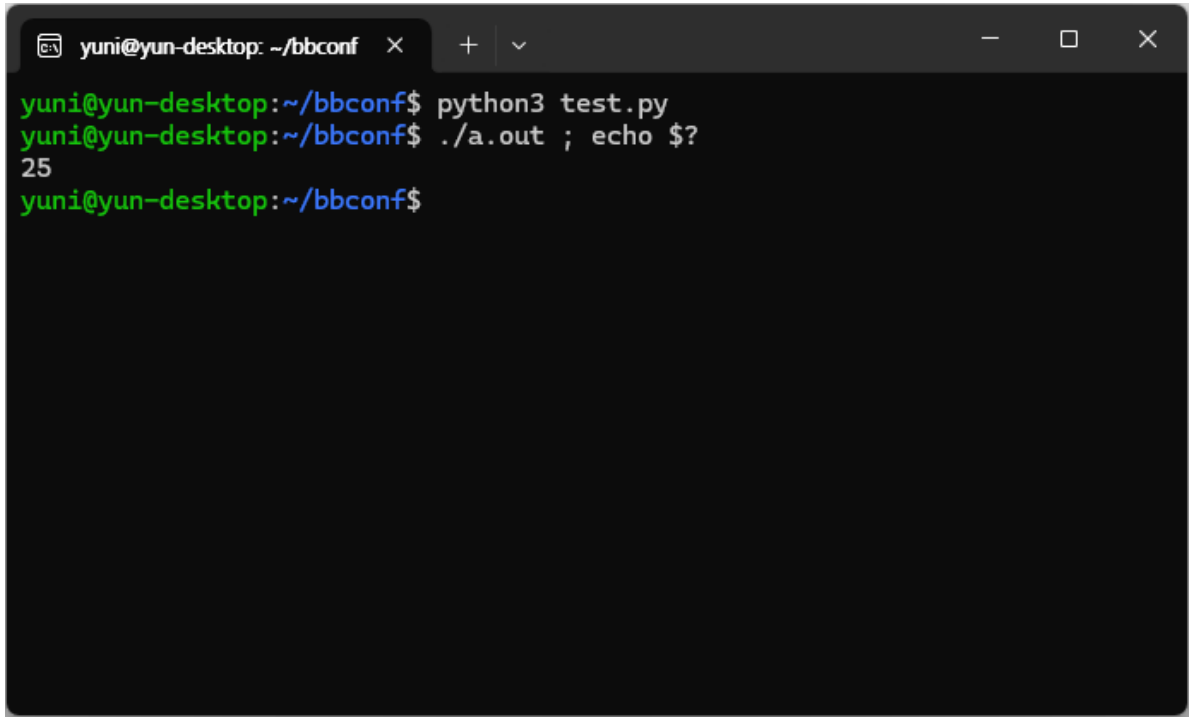
Best record **4320** Bytes

Header의 필요성



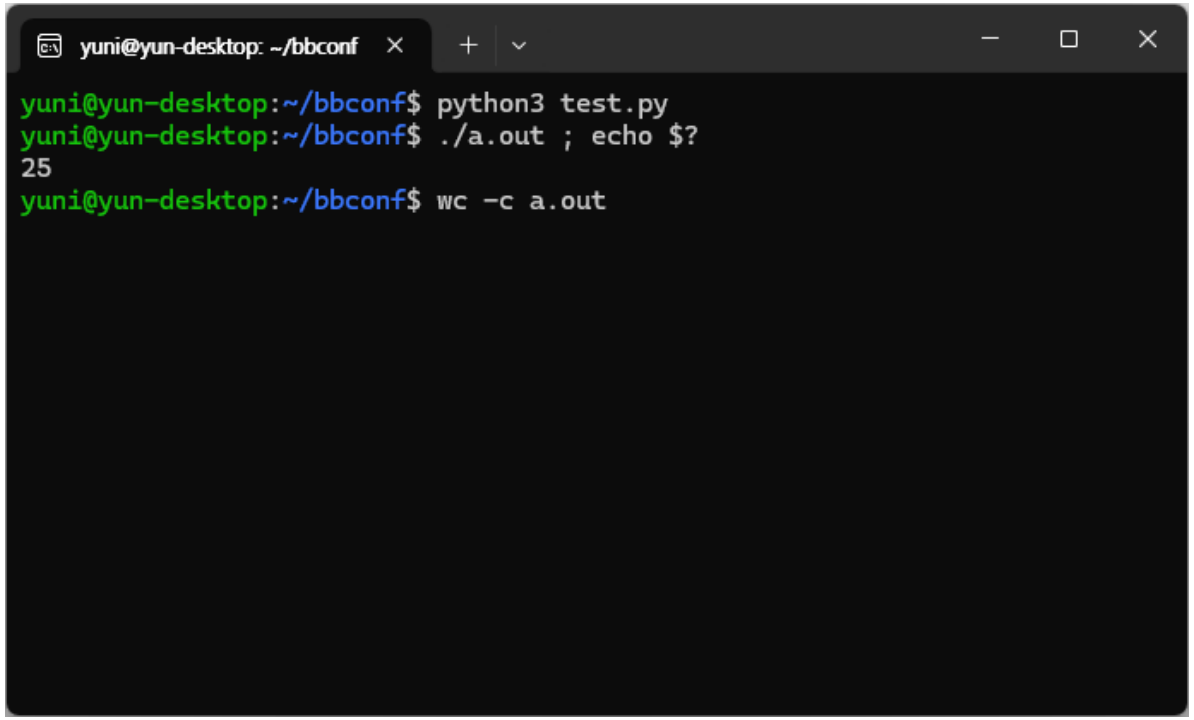
Best record **4320** Bytes

Header의 필요성



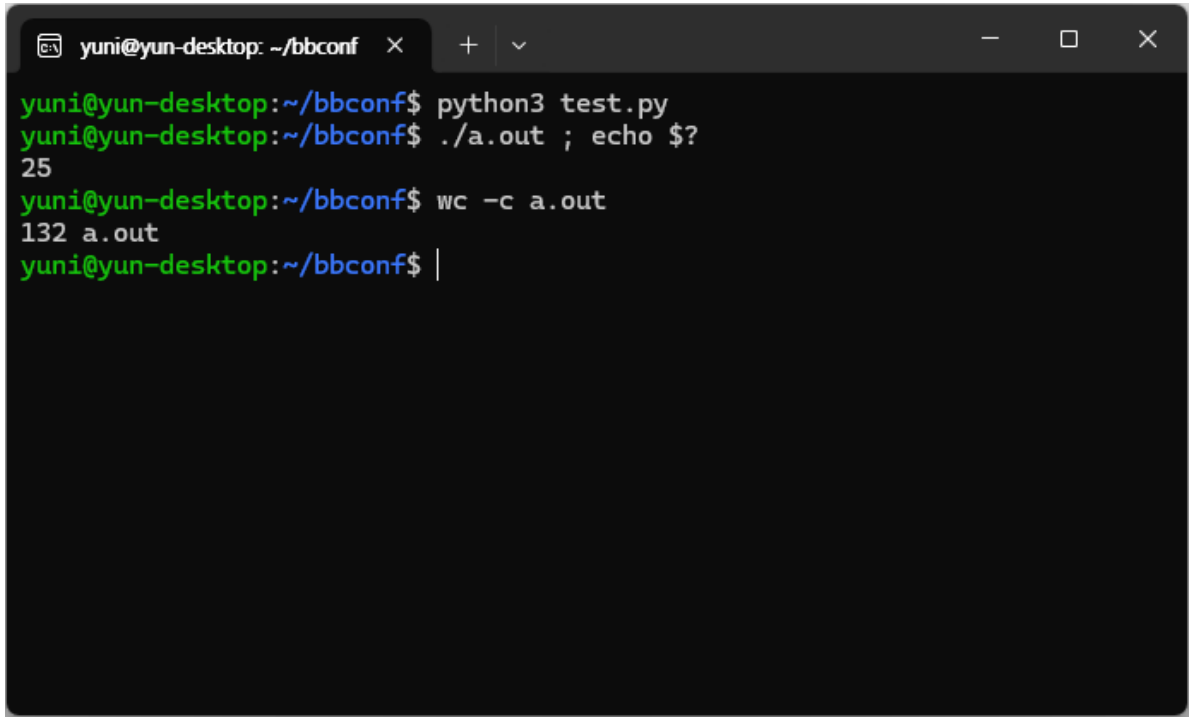
Best record **4320** Bytes

Header의 필요성



Best record **4320** Bytes

Header의 필요성

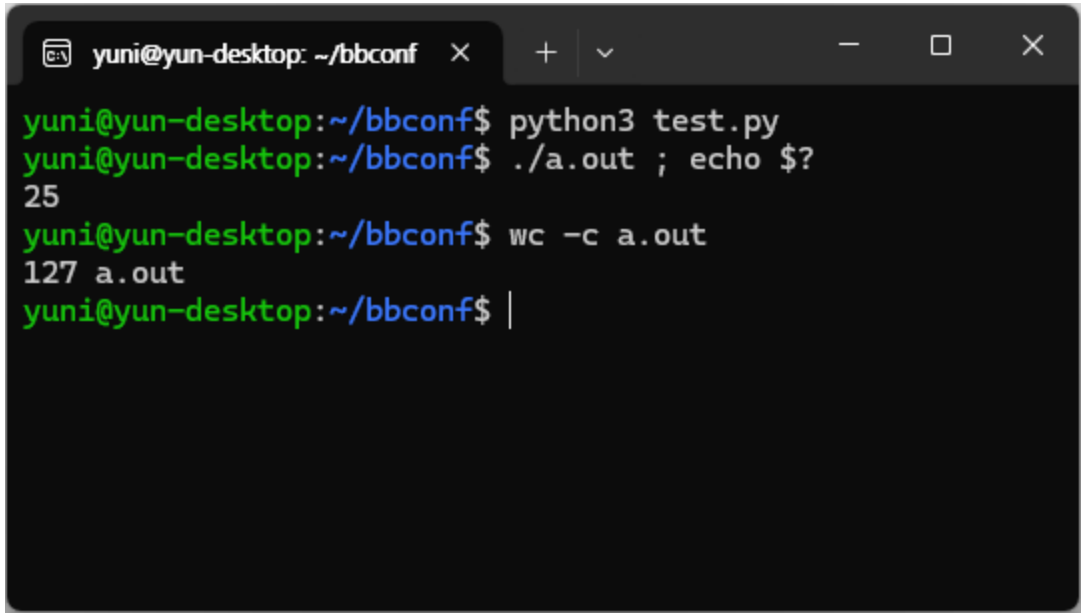


Best record **4320** -> **132** Bytes (▼4188)

또 다시 어셈블리 쇳코딩

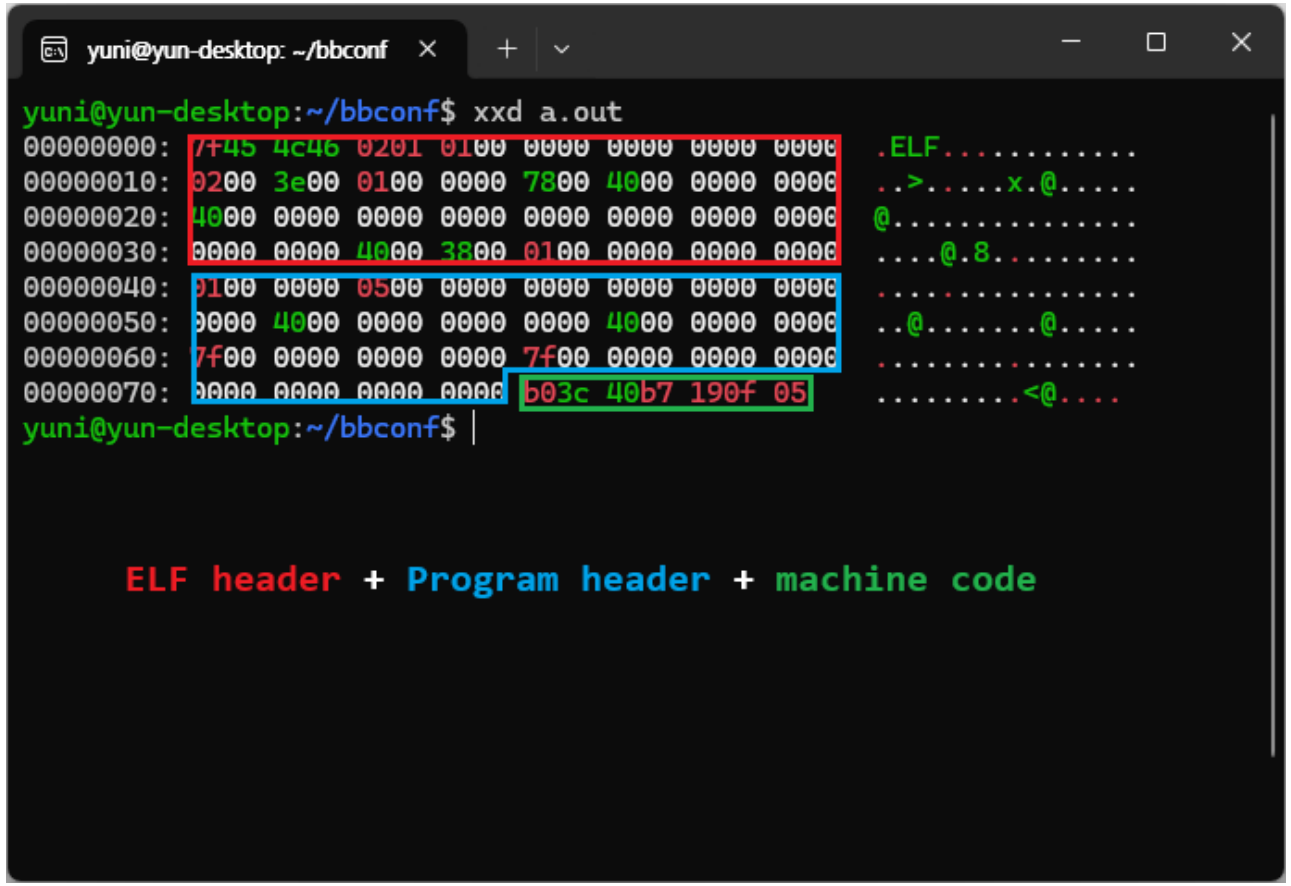
Best record **132** Bytes

또 다시 어셈블리 숏코딩



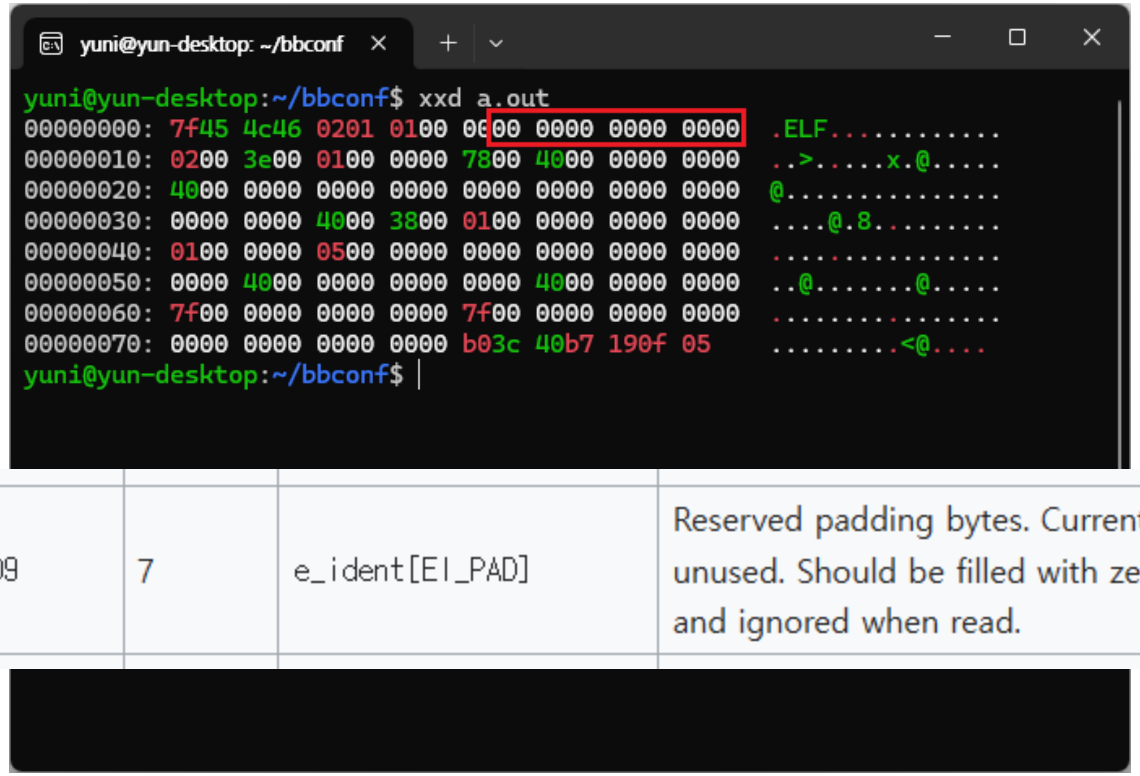
Best record **132** -> **127** Bytes (▼5)

더 줄일 수 있는가?



Best record **127** Bytes

더 줄일 수 있는가?



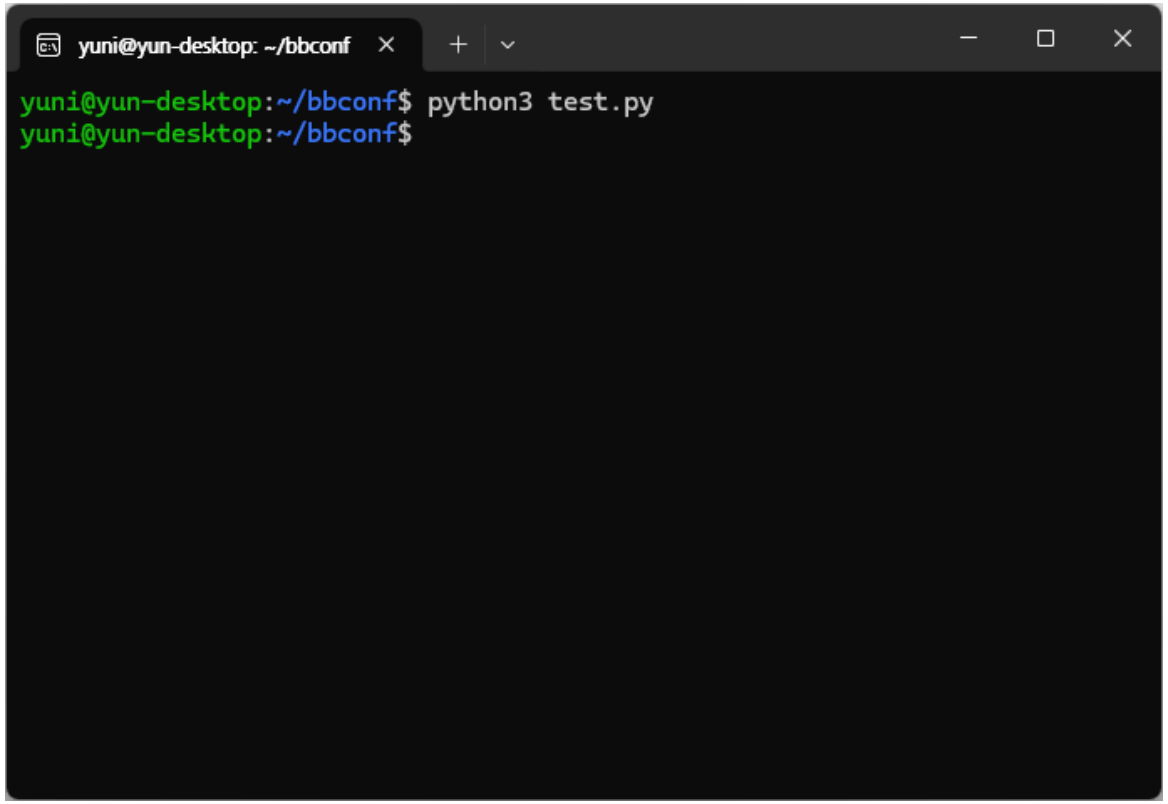
0x09	7	e_ident[EI_PAD]	Reserved padding bytes. Currently unused. Should be filled with zeros and ignored when read.

Best record **127** Bytes

더 줄일 수 있는가?

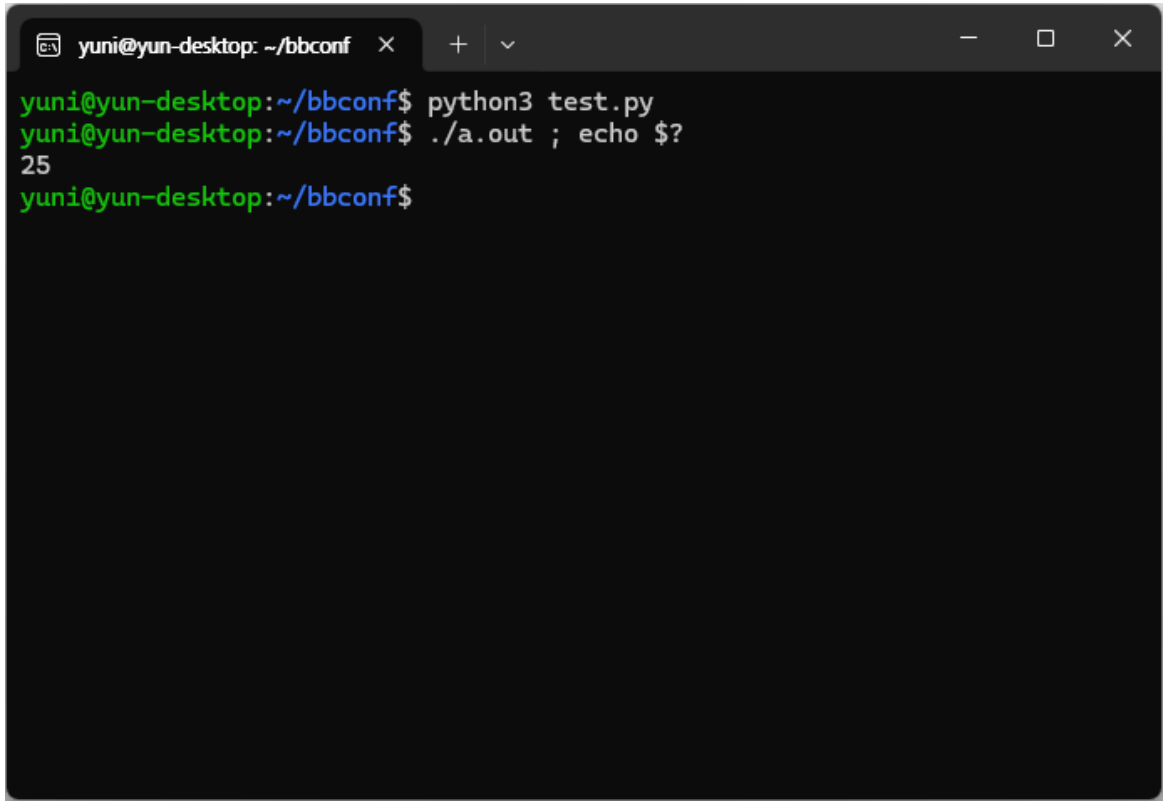
Best record **127** Bytes

더 줄일 수 있는가?



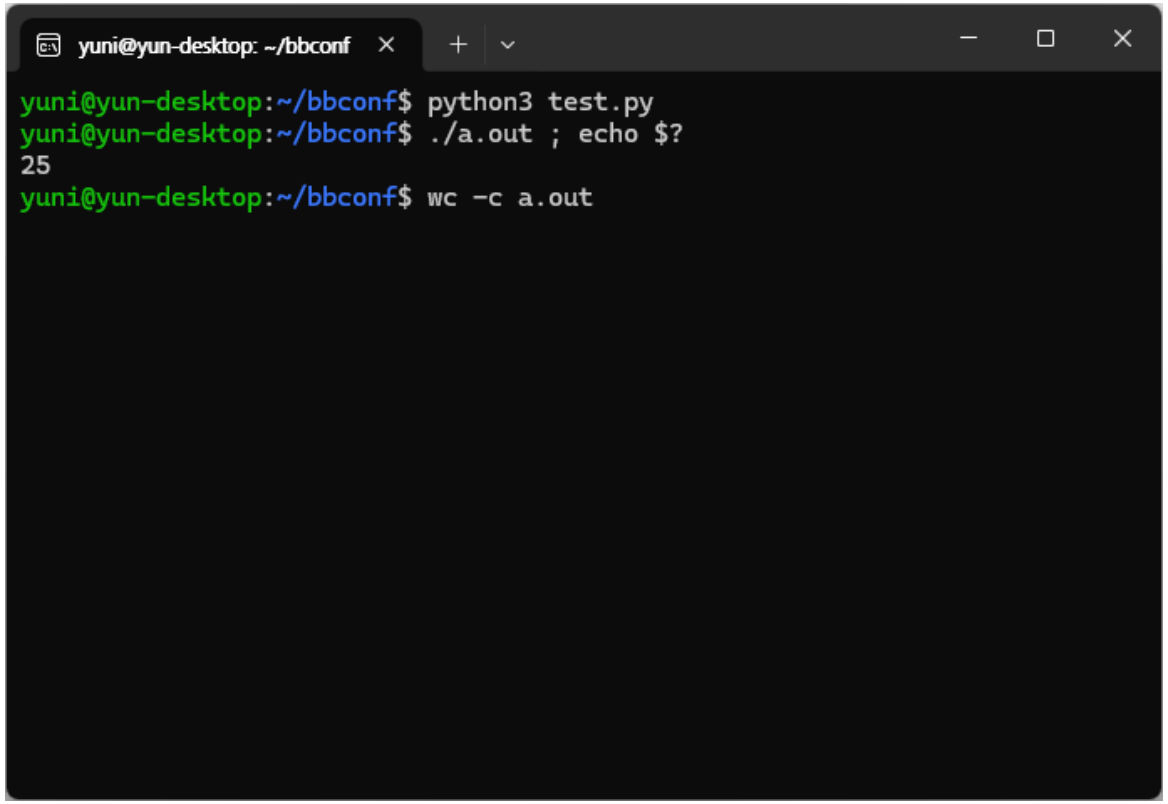
Best record **127** Bytes

더 줄일 수 있는가?



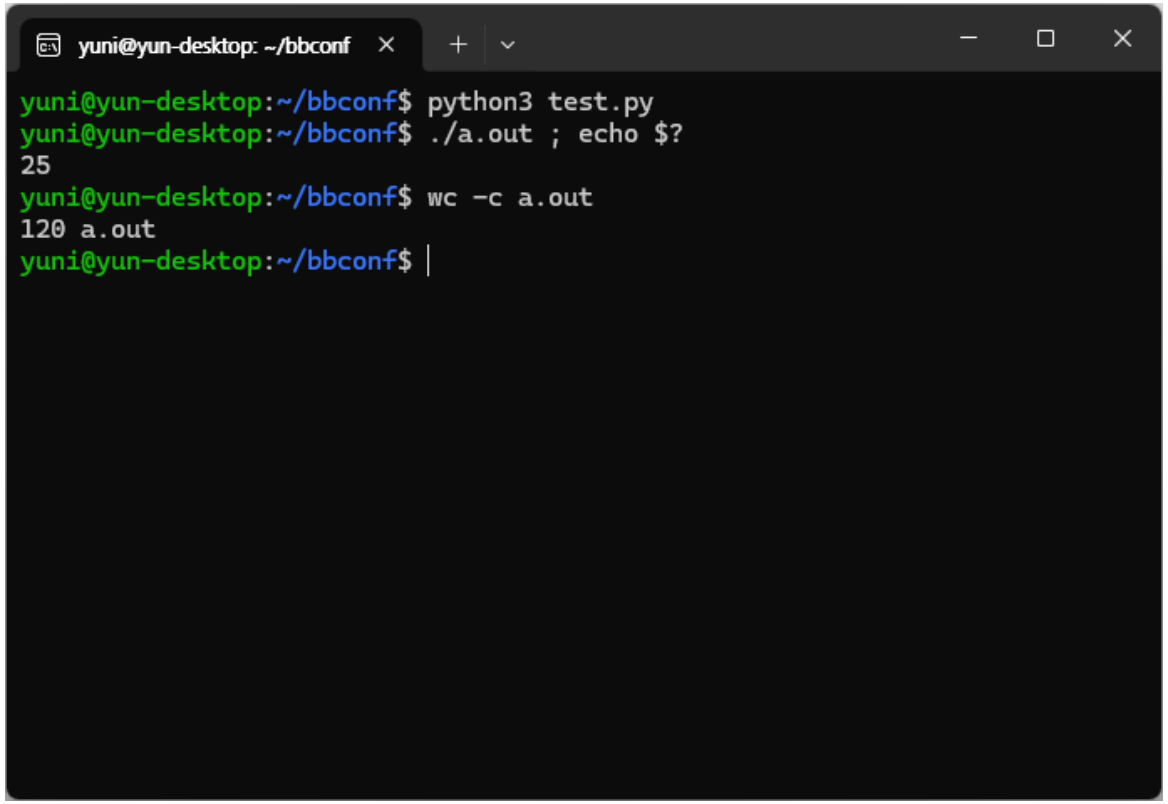
Best record **127** Bytes

더 줄일 수 있는가?



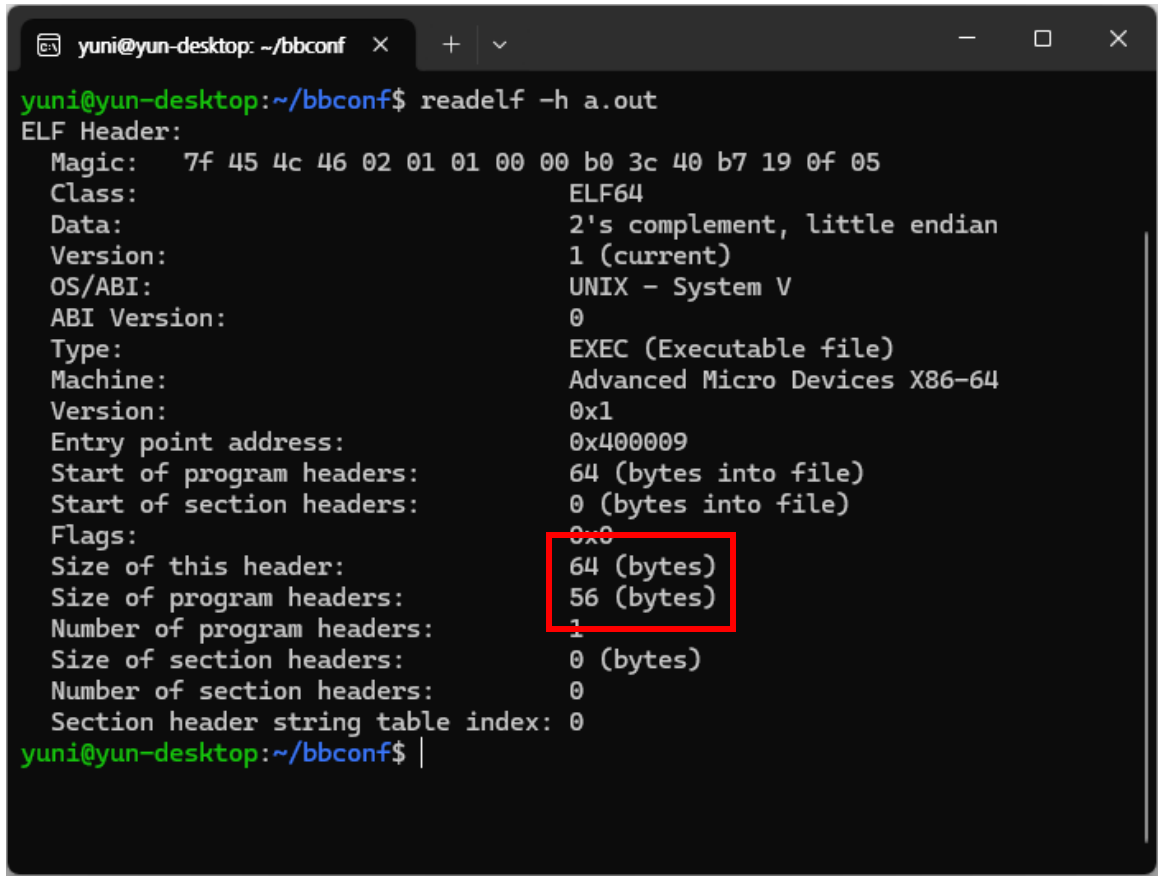
Best record **127** Bytes

더 줄일 수 있는가?



Best record **127** -> **120** Bytes (▼7)

한계에 도달



Best record **120** Bytes

한계에 도달

아직 **꼼수**가 남아있다

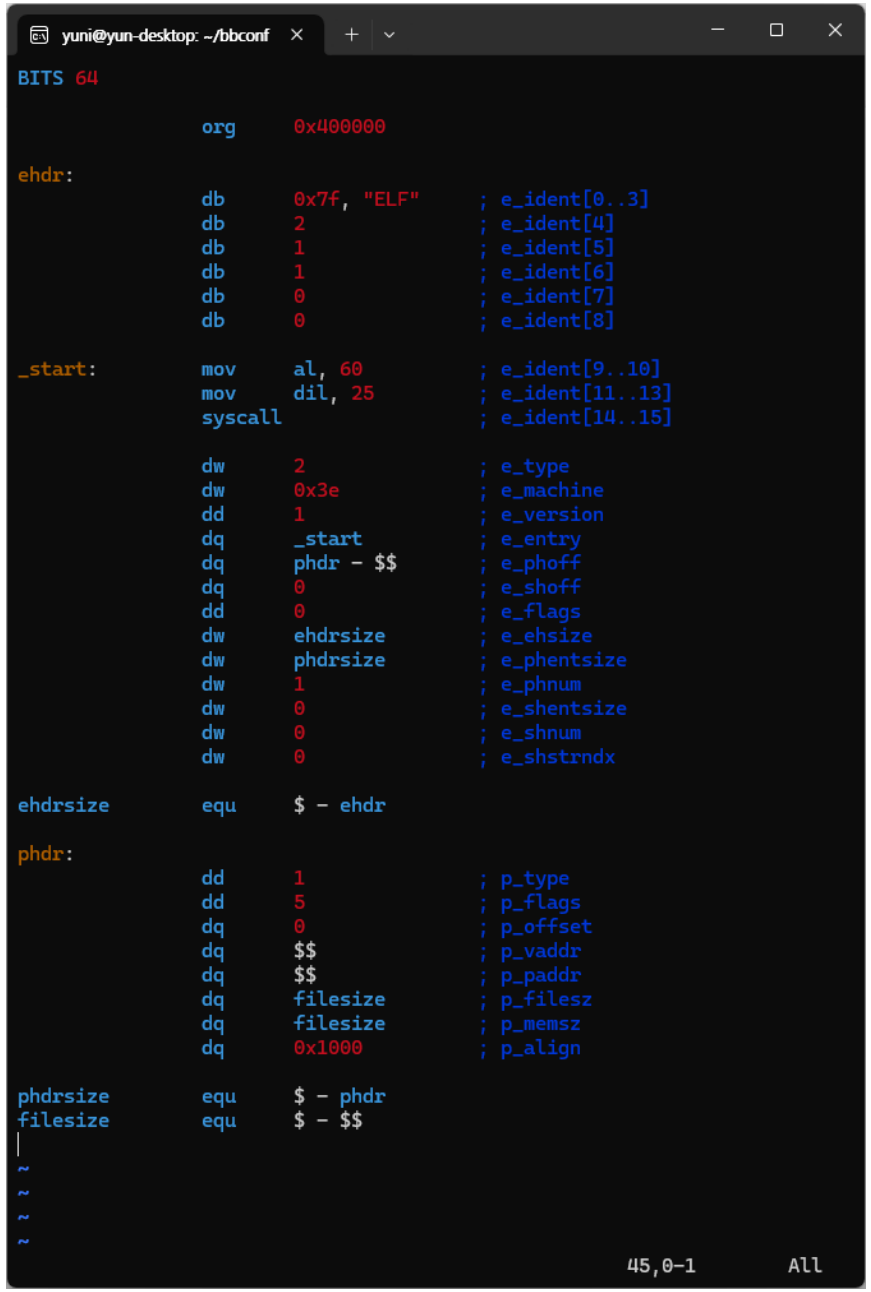
Best record **120** Bytes

nasm으로 bin 만들기

하지만 그 전에...
계속 바이너리를 들여다 보고 있으려니
눈이 빠질 것 같아서 다른 방법을 모색

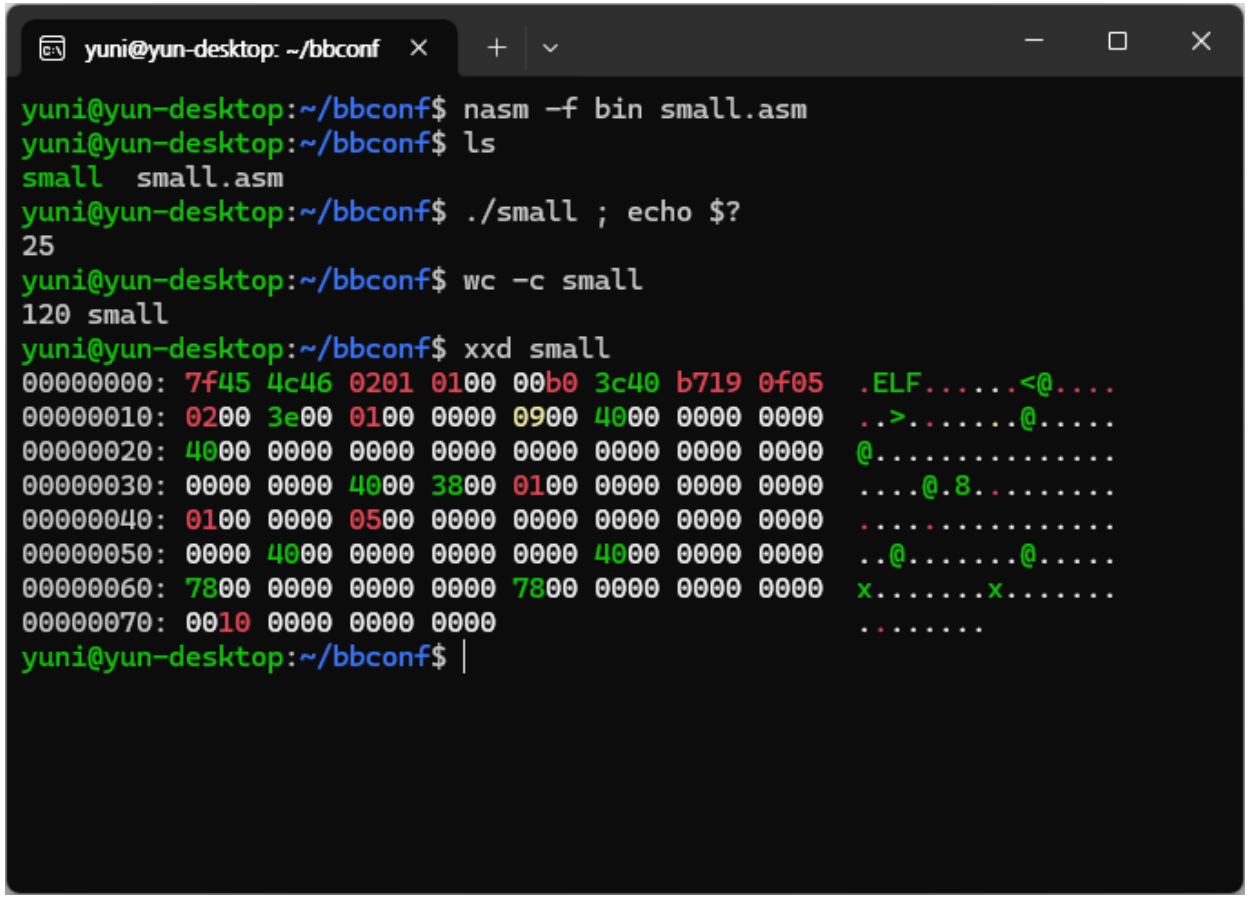
Best record **120** Bytes

nasm으로 bin 만들기



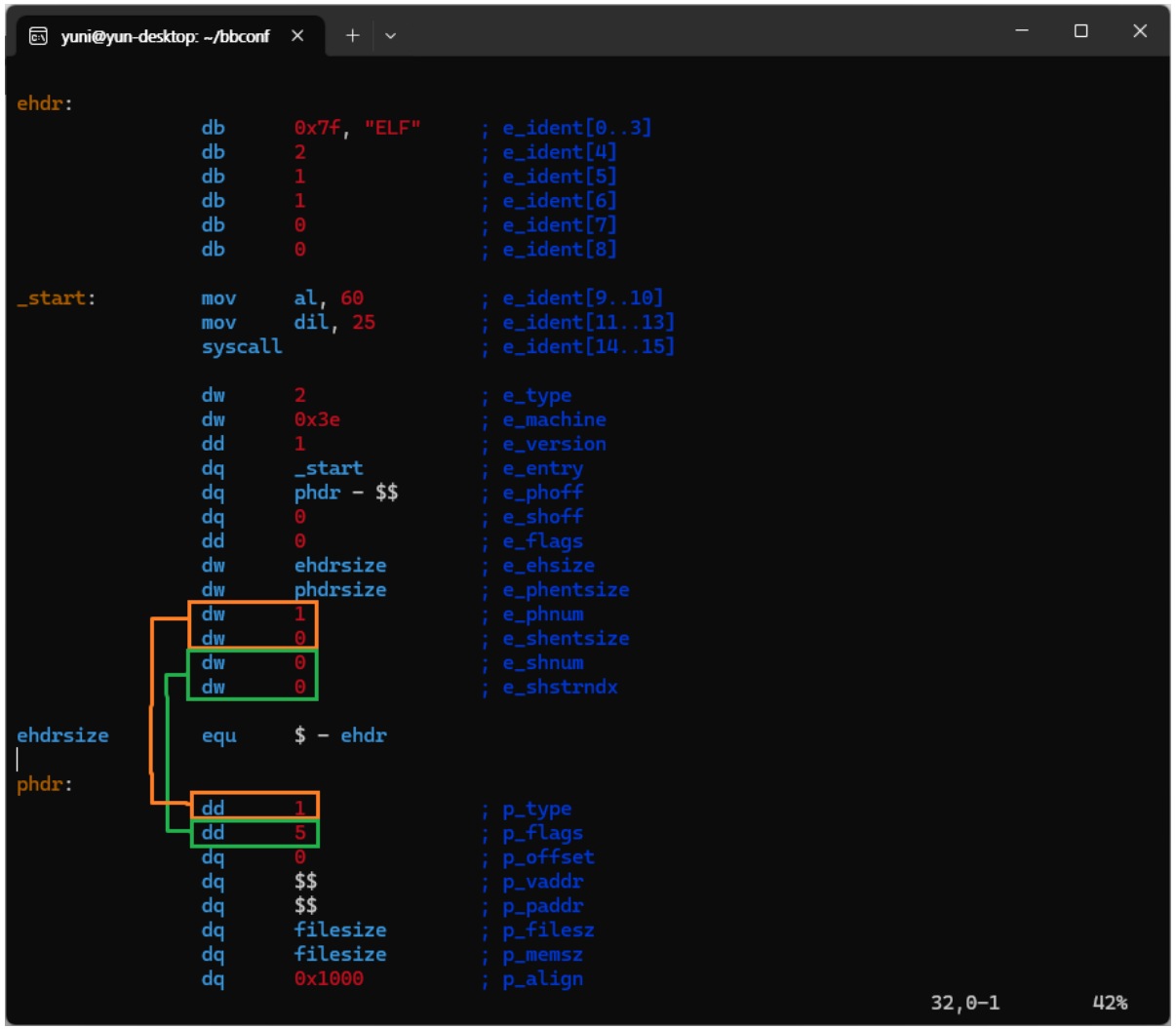
Best record **120** Bytes

nasm으로 bin 만들기



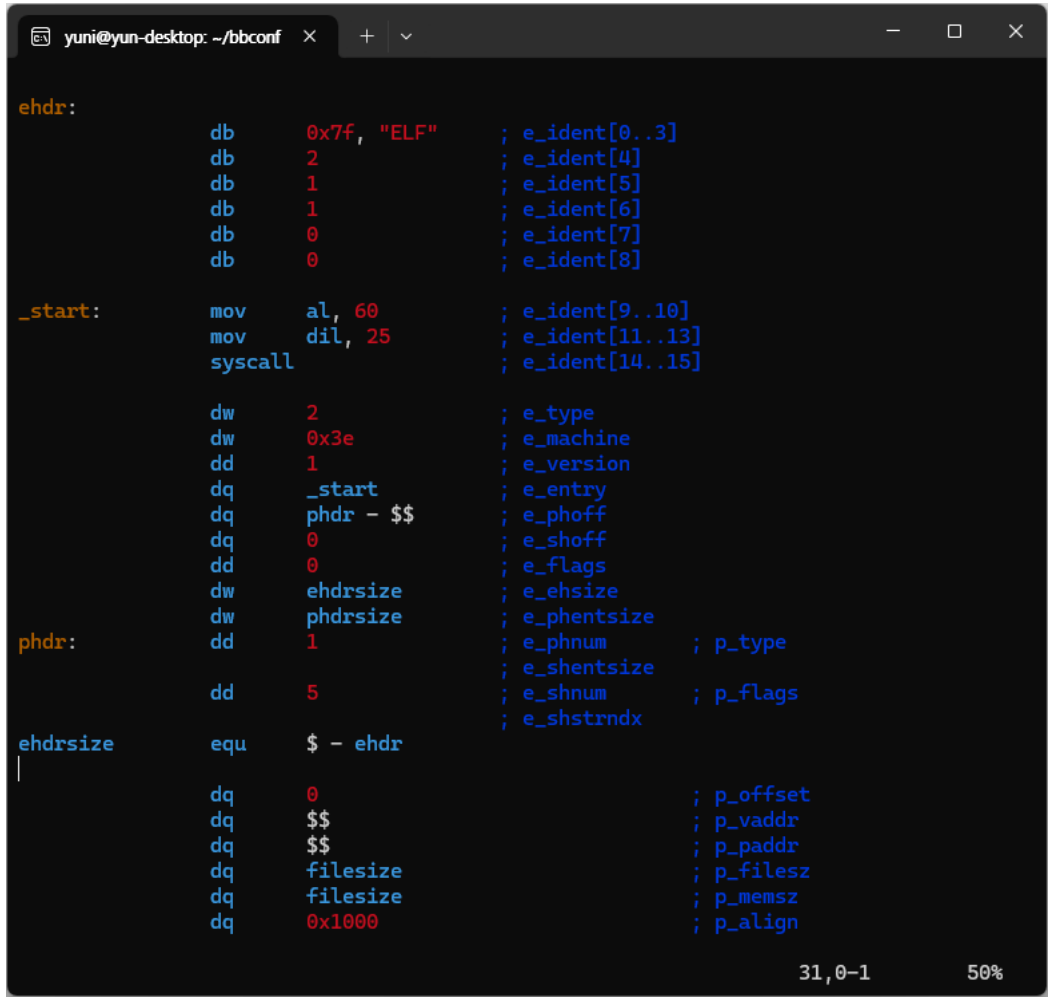
Best record **120** Bytes

어떤 꼼수가 있을까



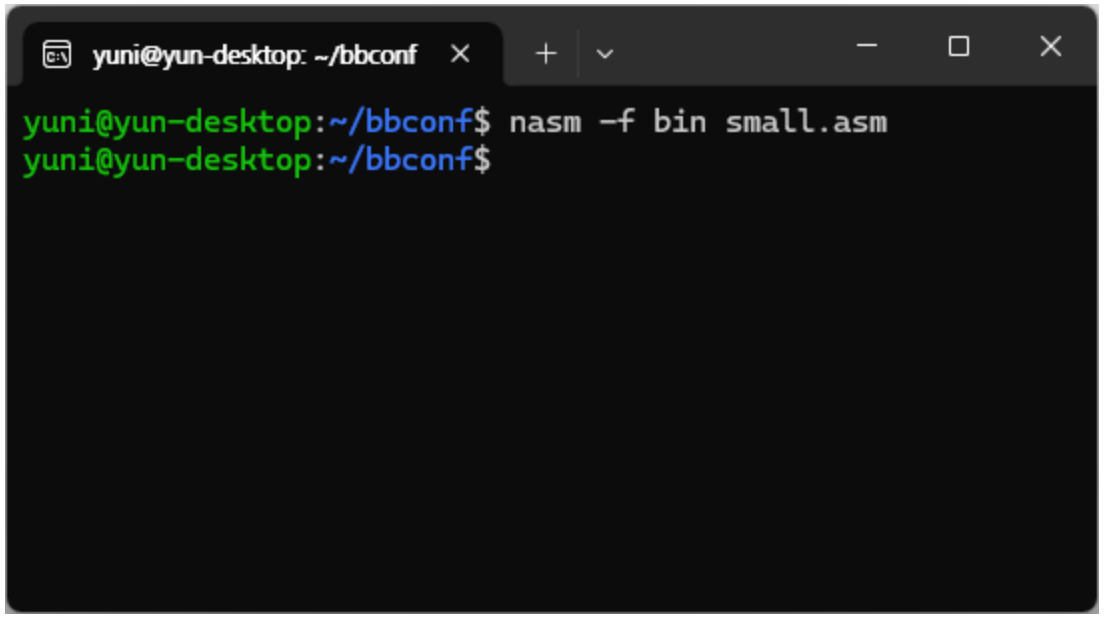
Best record **120** Bytes

어떤 꼼수가 있을까



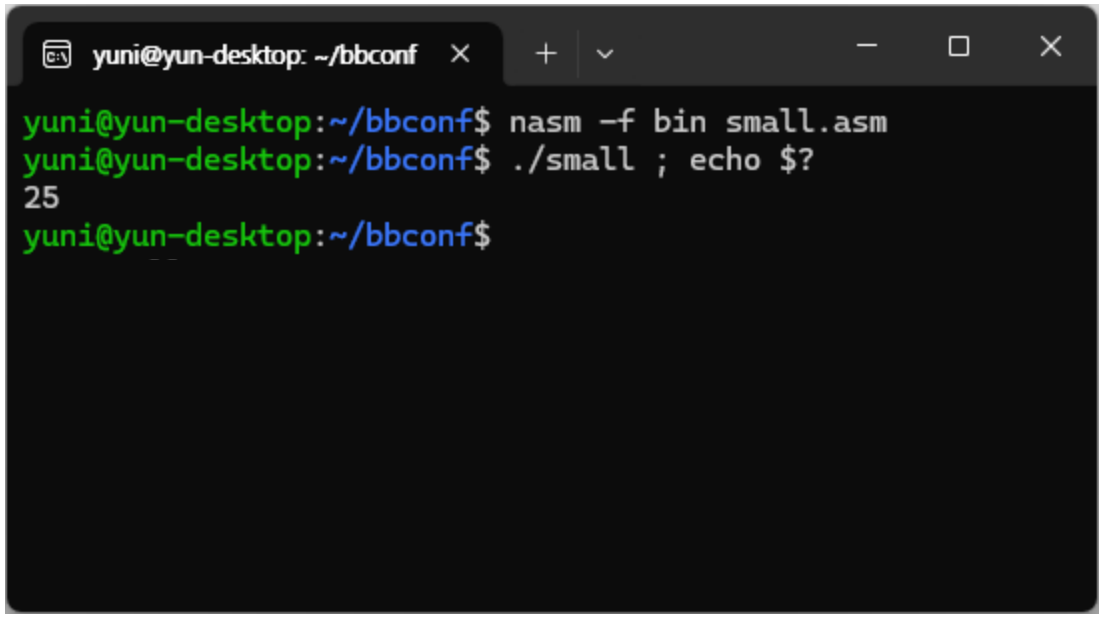
Best record **120** Bytes

어떤 꼼수가 있을까



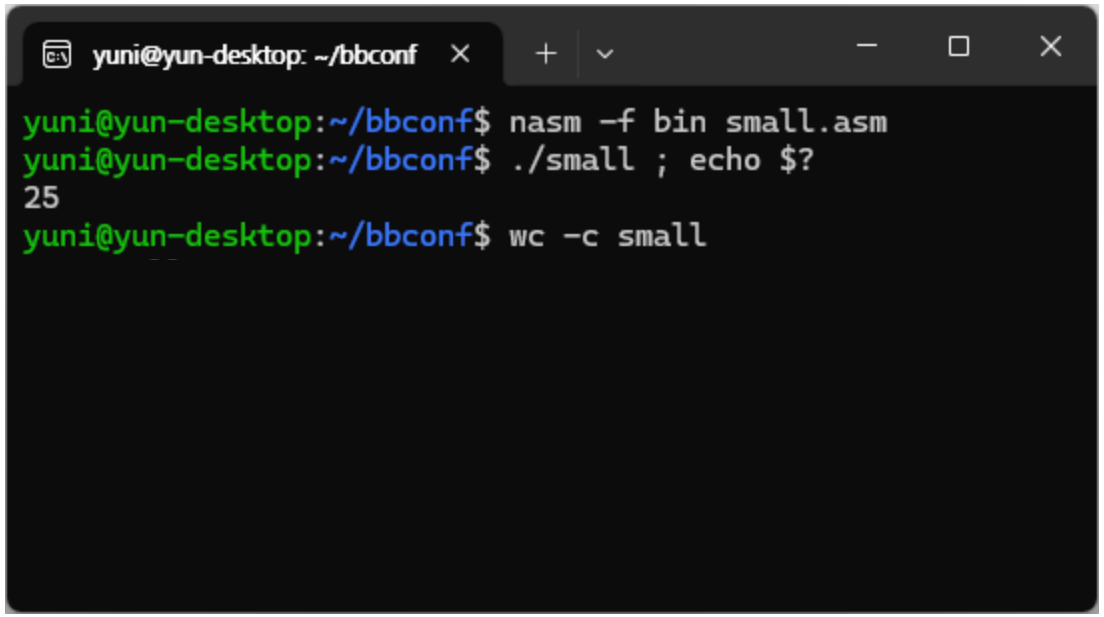
Best record **120** Bytes

어떤 꼼수가 있을까



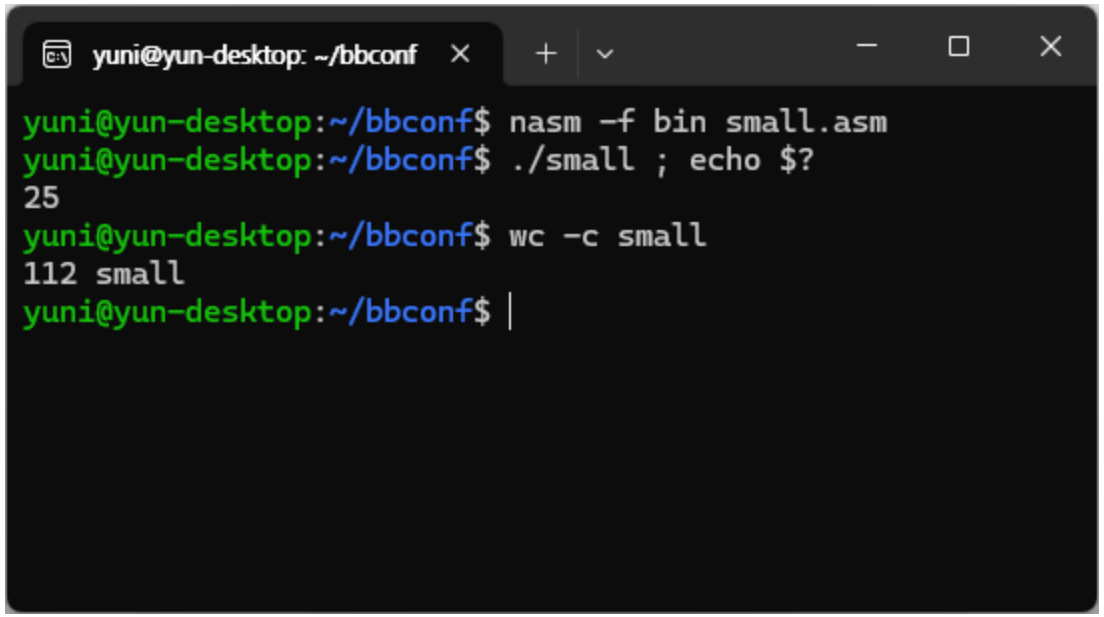
Best record **120** Bytes

어떤 꼼수가 있을까



Best record **120** Bytes

어떤 꼼수가 있을까



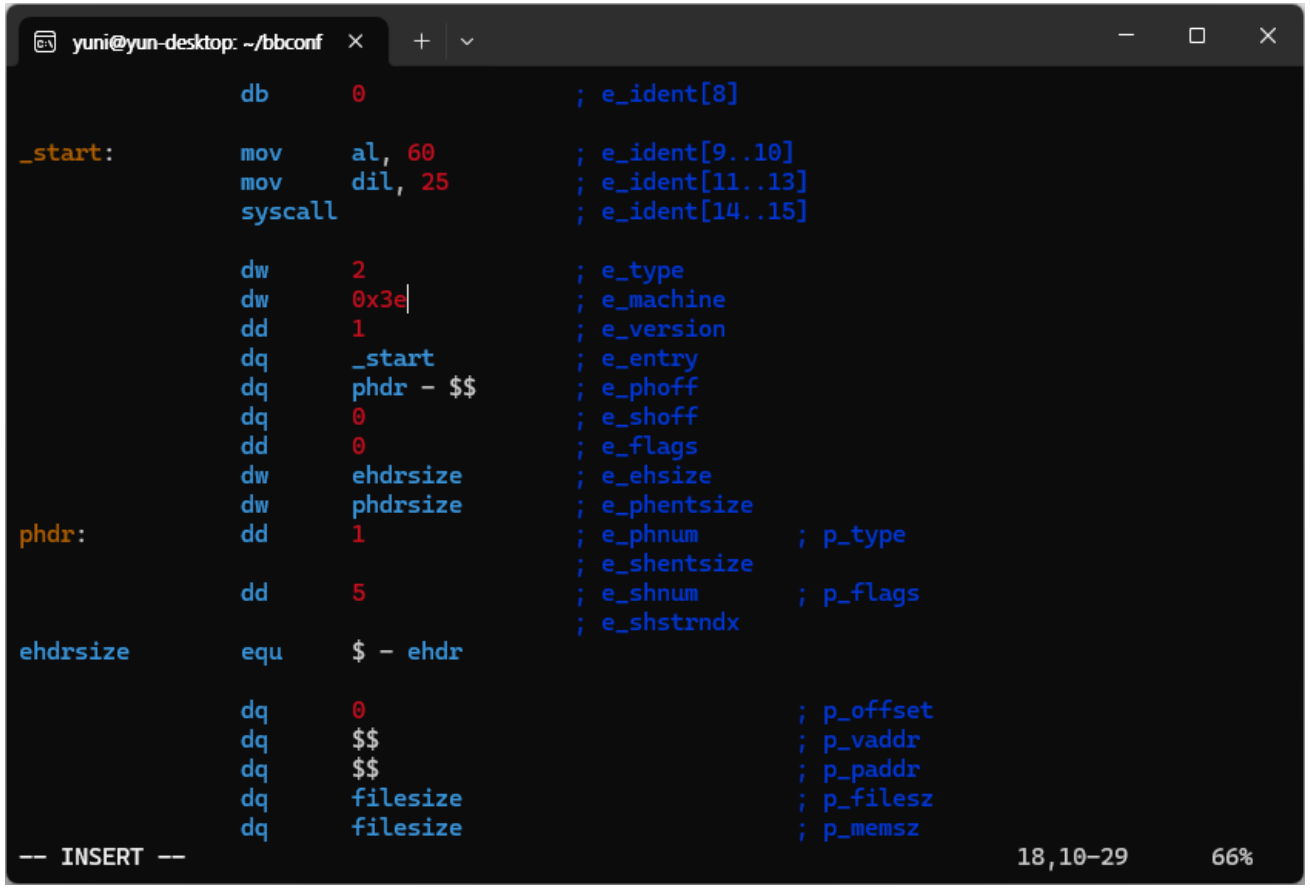
Best record **120** -> **112** Bytes (▼8)

활용 가능한 헤더의 필드들

스펙에서는 필드마다 요구사항이 있다.
하지만, 실제로 모든 필드들을 확인할까?

Best record **112** Bytes

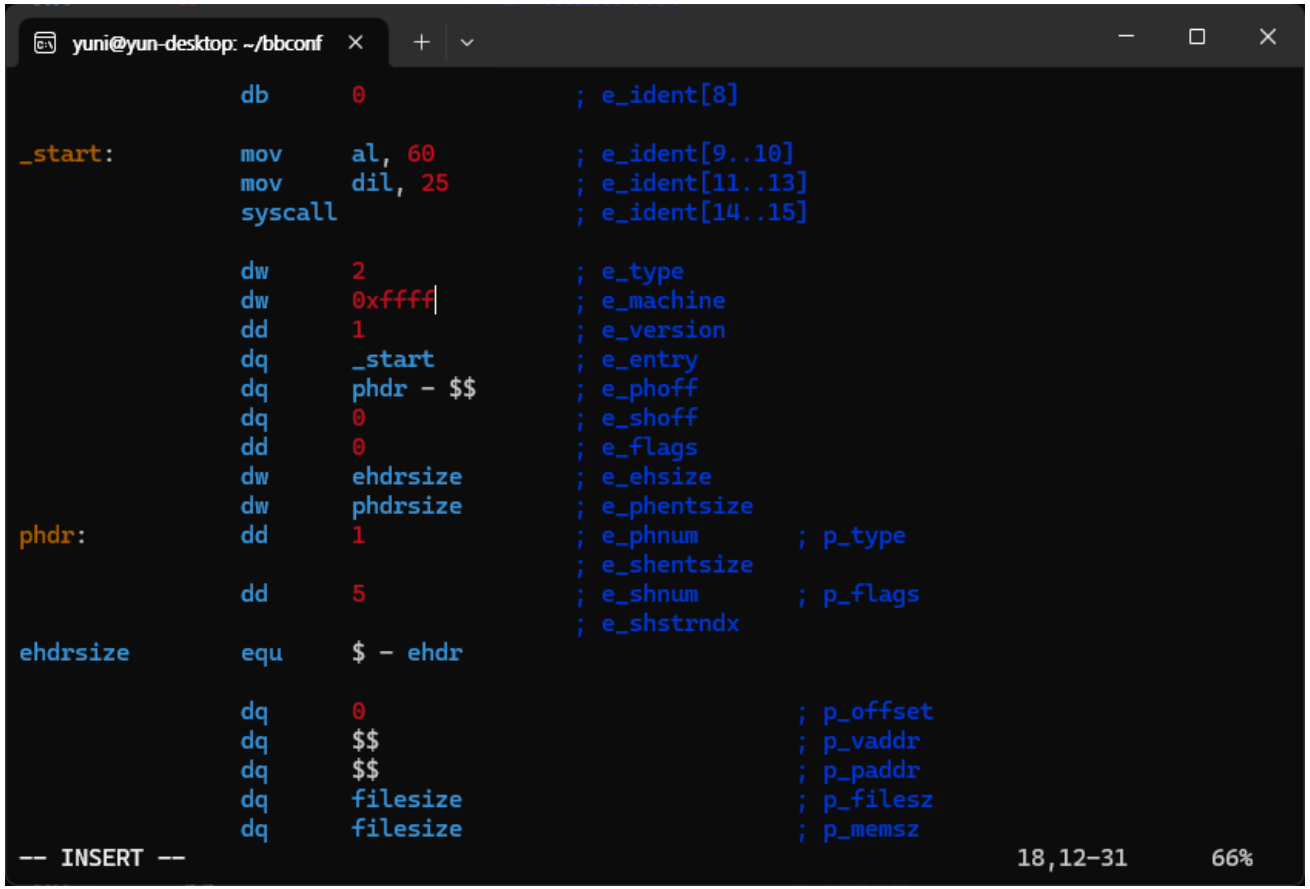
활용 가능한 헤더의 필드들



e_machine 필드는 0x3e가 AMD x86-64를 의미함

Best record **112** Bytes

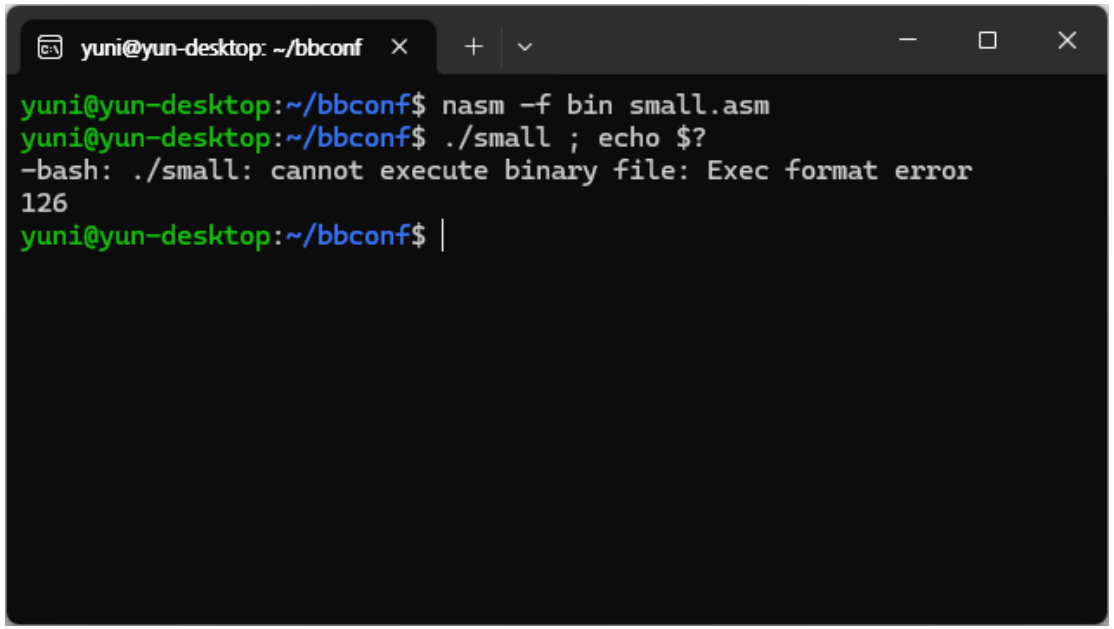
활용 가능한 헤더의 필드들



e_machine 필드는 0x3e가 AMD x86-64를 의미함
아무런 값이나 써도 정상 동작하는지 확인해 보자

Best record **112** Bytes

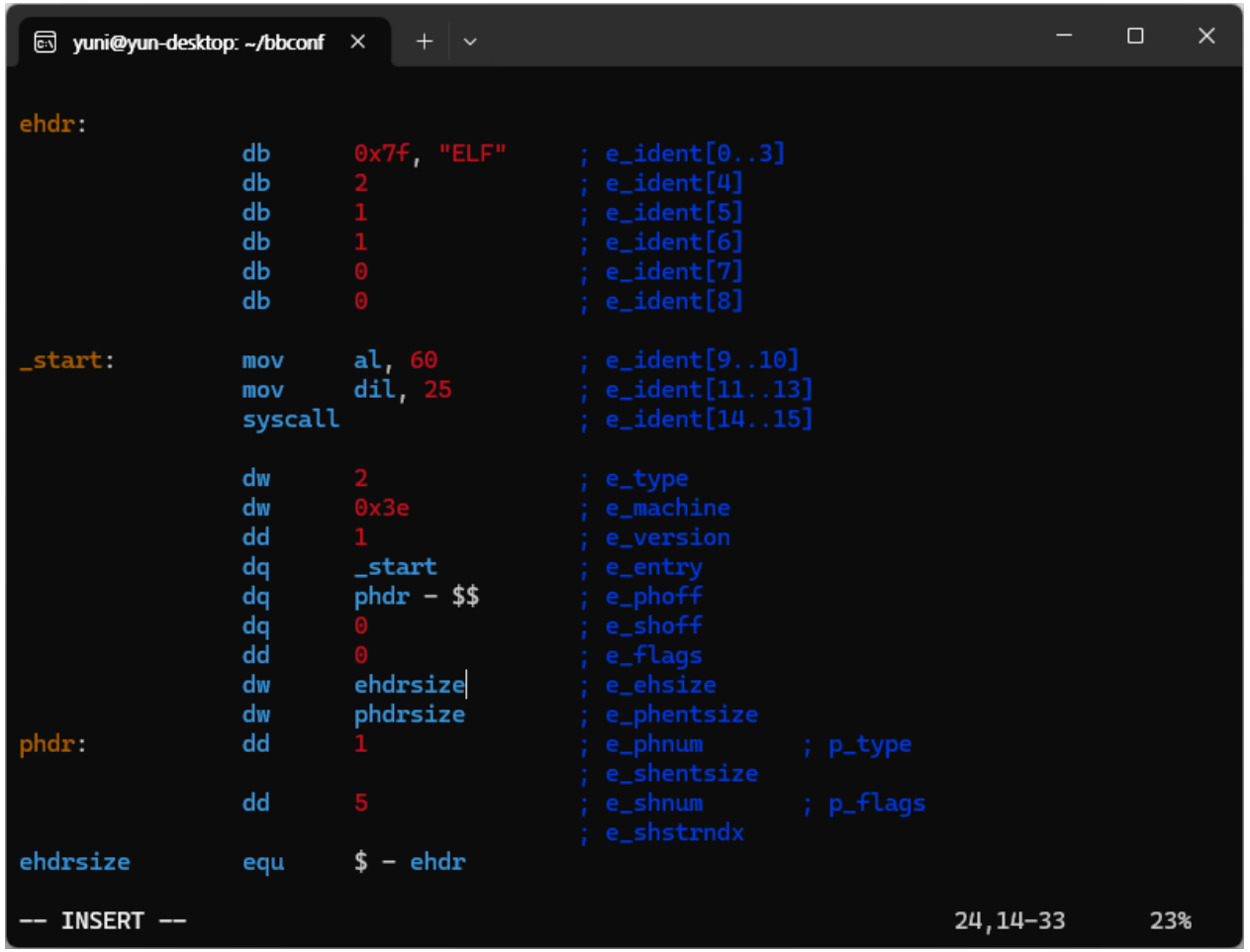
활용 가능한 헤더의 필드들



실패...

Best record **112** Bytes

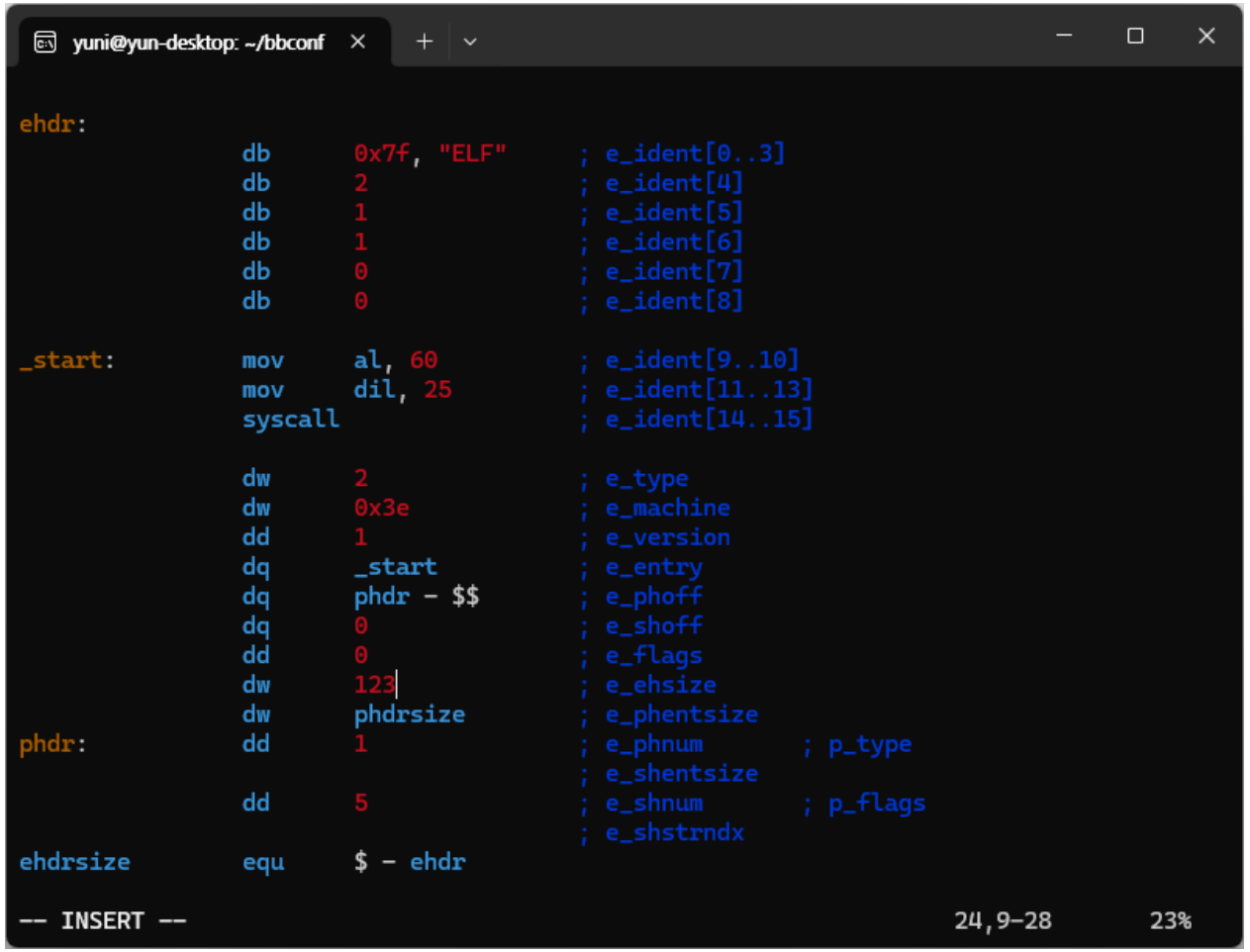
활용 가능한 헤더의 필드들



e_ehsize	This member holds the ELF header's size in bytes.

Best record **112** Bytes

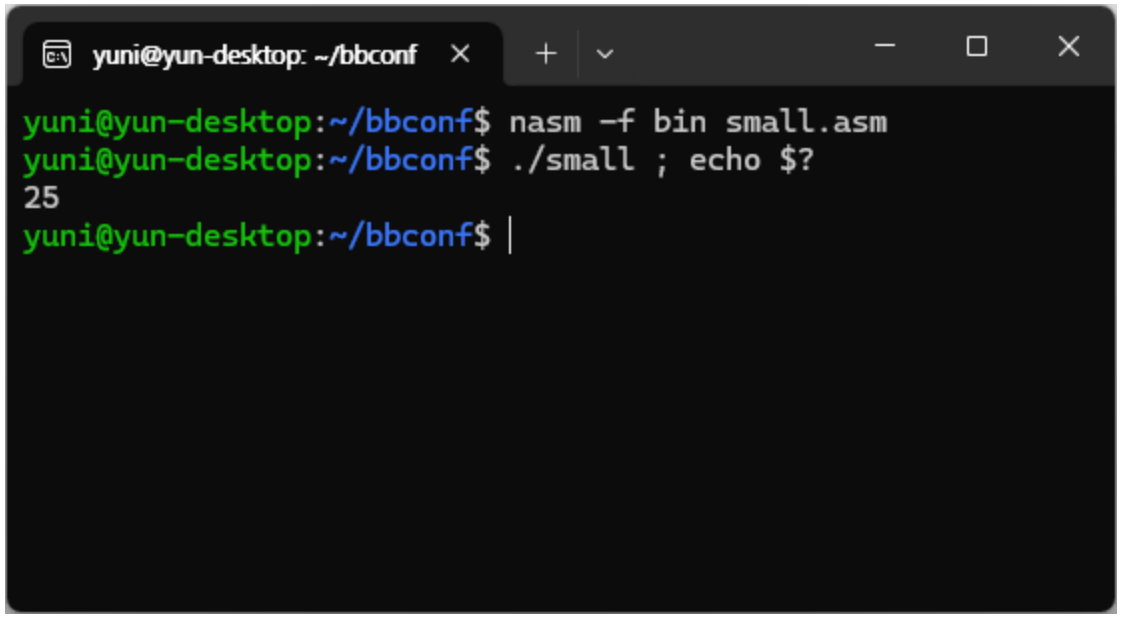
활용 가능한 헤더의 필드들



`e_ehsize` This member holds the ELF header's size in bytes.

Best record **112** Bytes

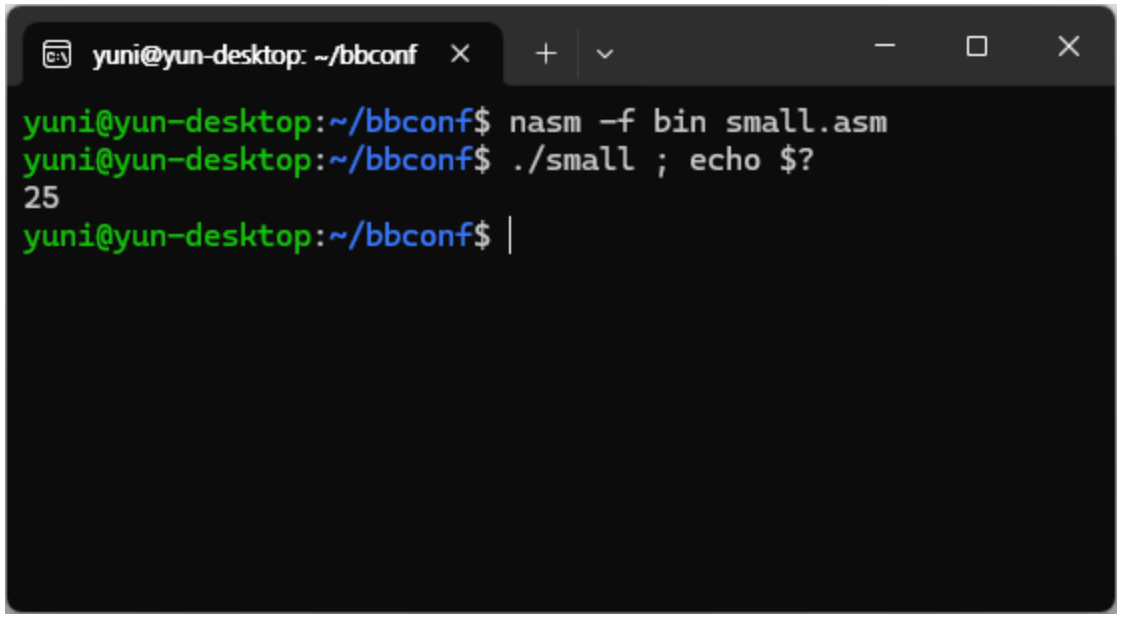
활용 가능한 헤더의 필드들



성공!!!

Best record **112** Bytes

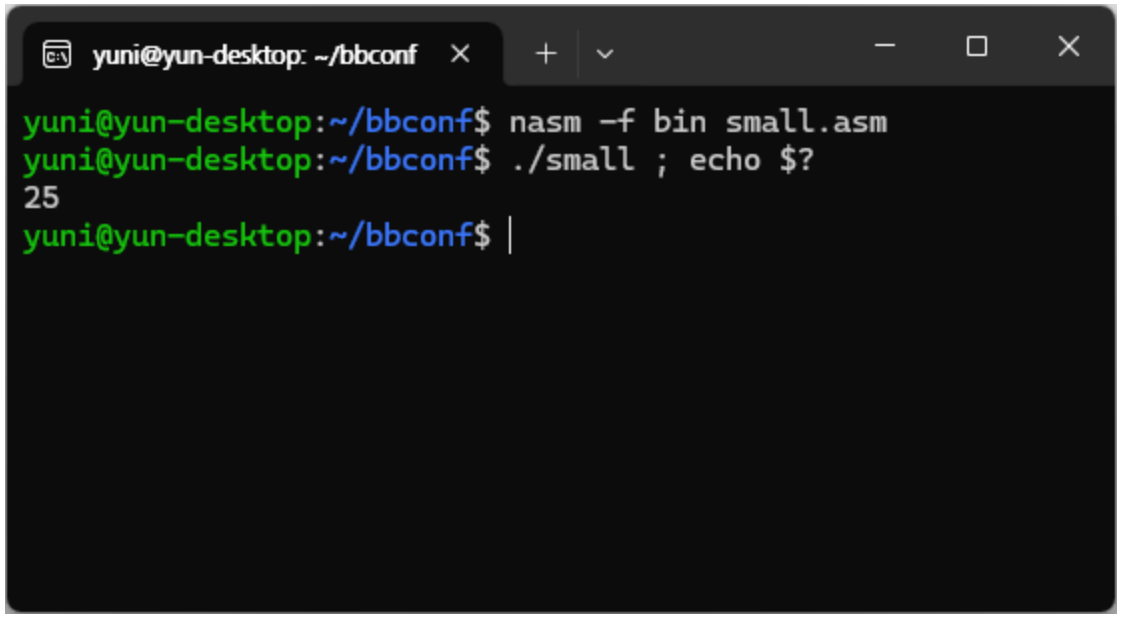
활용 가능한 헤더의 필드들



성공!!!
1처럼 작은 값을 줘도 잘 동작하는 것을 확인

Best record **112** Bytes

활용 가능한 헤더의 필드들



성공!!!
1처럼 작은 값을 줘도 잘 동작하는 것을 확인
e_ehsize는 활용 가능한 필드

Best record **112** Bytes

활용 가능한 헤더의 필드들

3. File header

The file header is located at the beginning of the file, and is used to locate the other parts of the file. The structure is shown in Figure 2.

```
typedef struct
{
        unsigned char   e_ident[16];    /* ELF identification */
        Elf64_Half      e_type;         /* Object file type */
        Elf64_Half      e_machine;      /* Machine type */
        Elf64_Word      e_version;      /* Object file version */
        Elf64_Addr      e_entry;        /* Entry point address */
        Elf64_Off       e_phoff;        /* Program header offset */
        Elf64_Off       e_shoff;        /* Section header offset */
        Elf64_Word      e_flags;        /* Processor-specific flags */
        Elf64_Half      e_ehsize;       /* ELF header size */
        Elf64_Half      e_phentsize;    /* Size of program header entry */
        Elf64_Half      e_phnum;        /* Number of program header entries */
        Elf64_Half      e_shentsize;    /* Size of section header entry */
        Elf64_Half      e_shnum;        /* Number of section header entries */
        Elf64_Half      e_shstrndx;     /* Section name string table index */
} Elf64_Ehdr;
```

4 + 12

Figure 2. ELF-64 Header

8. Program header table

In executable and shared object files, sections are grouped into segments for loading. The program header table contains a list of entries describing each segment. The structure of the program header table entry is shown in Figure 6.

```
typedef struct
{
        Elf64_Word      p_type;         /* Type of segment */
        Elf64_Word      p_flags;        /* Segment attributes */
        Elf64_Off       p_offset;       /* Offset in file */
        Elf64_Addr      p_vaddr;        /* Virtual address in memory */
        Elf64_Addr      p_paddr;        /* Reserved */
        Elf64_Xword     p_filesz;       /* Size of segment in file */
        Elf64_Xword     p_memsz;        /* Size of segment in memory */
        Elf64_Xword     p_align;        /* Alignment of segment */
} Elf64_Phdr;
```

Figure 6. ELF-64 Program Header Table Entry

Best record **112** Bytes

활용 가능한 헤더의 필드들

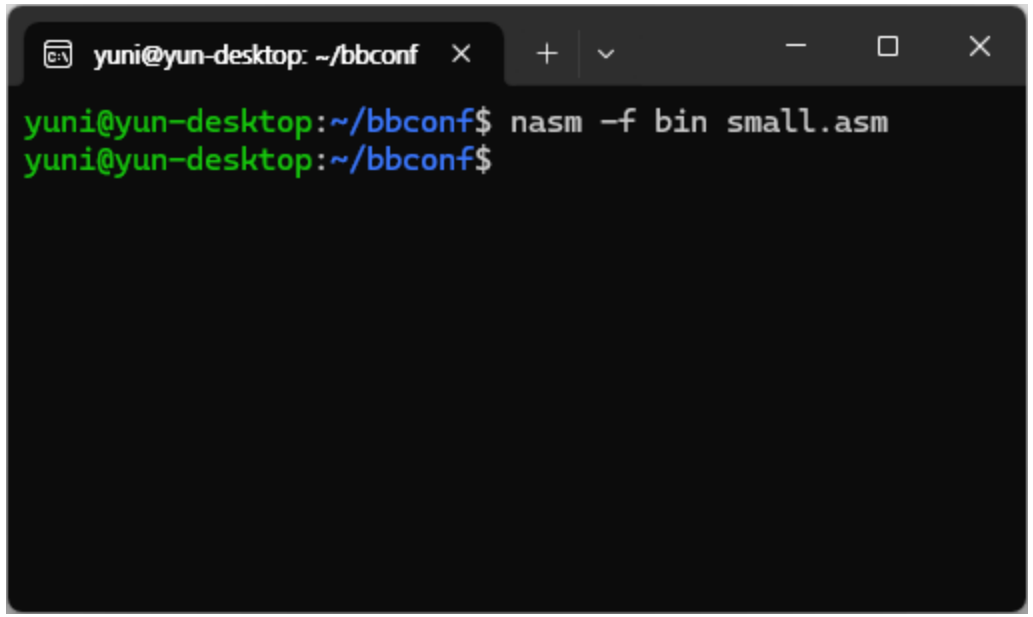
0x02	e_type		
0x00			
0x3e	e_machine		
0x00			
	e_version		
	e_entry		
0x40	e_phoff		
0x00			
0x00			
0x00			
0x00			
0x00			
0x00			
0x00			
	e_shoff		
	e_flags		
		p_type	0x01
			0x00
			0x00
	e_ehsize		0x00
		p_flags	0x05
0x38	e_phentsize		
0x00			
0x01	e_phnum		
0x00		p_offset	0x00
	e_shentsize		0x00
			0x00
	e_shnum		0x00
			0x00
	e_shstrndx		0x00
			0x00
			0x00
		p_vaddr	

yuni@yun-desktop: ~/bbconf

```
BITS 64

                org     0x400000

                db      0x7f, "ELF"     ; e_ident[0..3]
                db      2, 1, 1, 0      ; e_ident[4..7]
                db      0               ; e_ident[8]
_start:         mov     al, 60          ; e_ident[9..10]
                mov     dil, 25         ; e_ident[11..13]
                syscall                 ; e_ident[14..15]
                dw      2               ; e_type
                dw      0x3e            ; e_machine
                dd      1               ; e_version
                dq      _start          ; e_entry
                dq      phdr - $$       ; e_phoff
                dq      321             ; e_shoff
                db      0               ; e_flags
phdr:           db      1                               ; p_type
                dw      0
                db      0               ; e_ehsize
                db      5                               ; p_flags
                dw      0x38            ; e_phentsize
                dw      1               ; e_phnum
                                                        ; p_offset
                dd      0
                dw      0
                db      0
                dq      $$                              ; p_vaddr
                dq      $$                              ; p_paddr
                dq      filesize                        ; p_filesz
                dq      filesize                        ; p_memsz
                dq      0x1000                          ; p_align

filesize        equ     $ - $$
~
~
                                                35,0-1        All
```

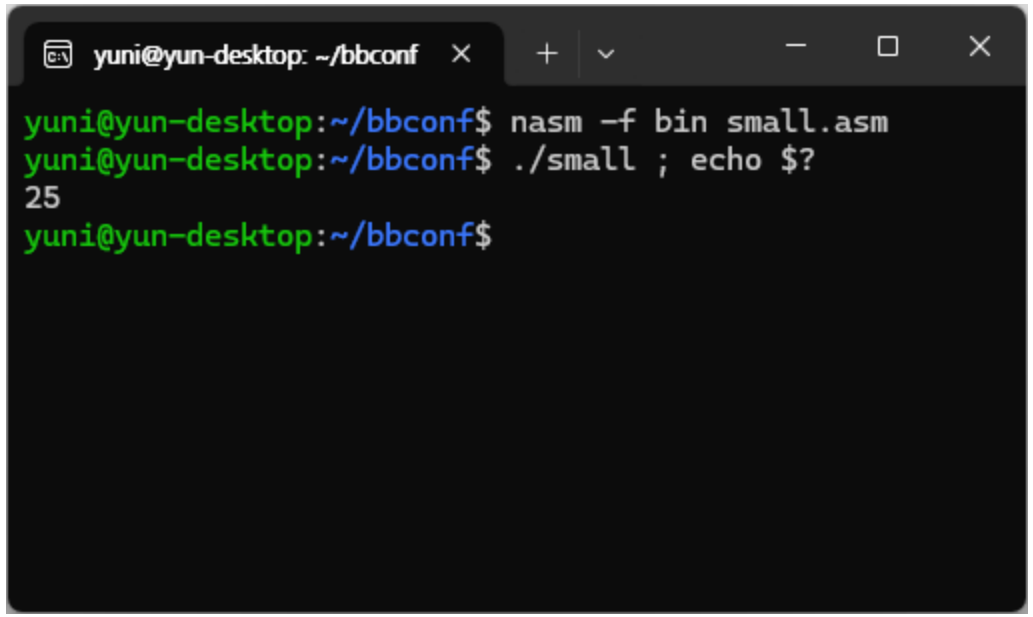
Best record **112** Bytes

활용 가능한 헤더의 필드들



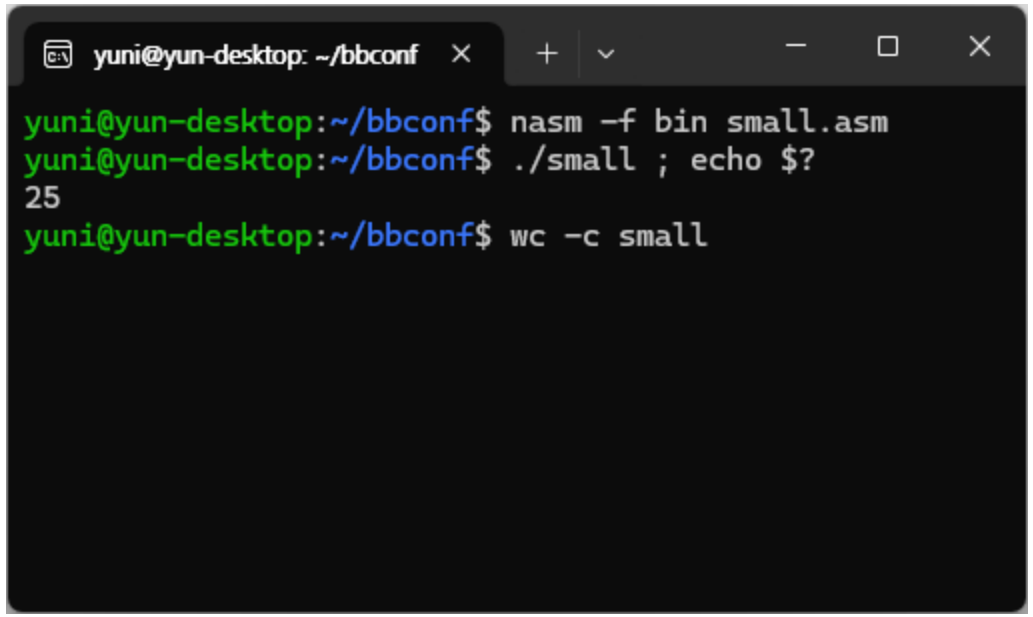
Best record **112** Bytes

활용 가능한 헤더의 필드들



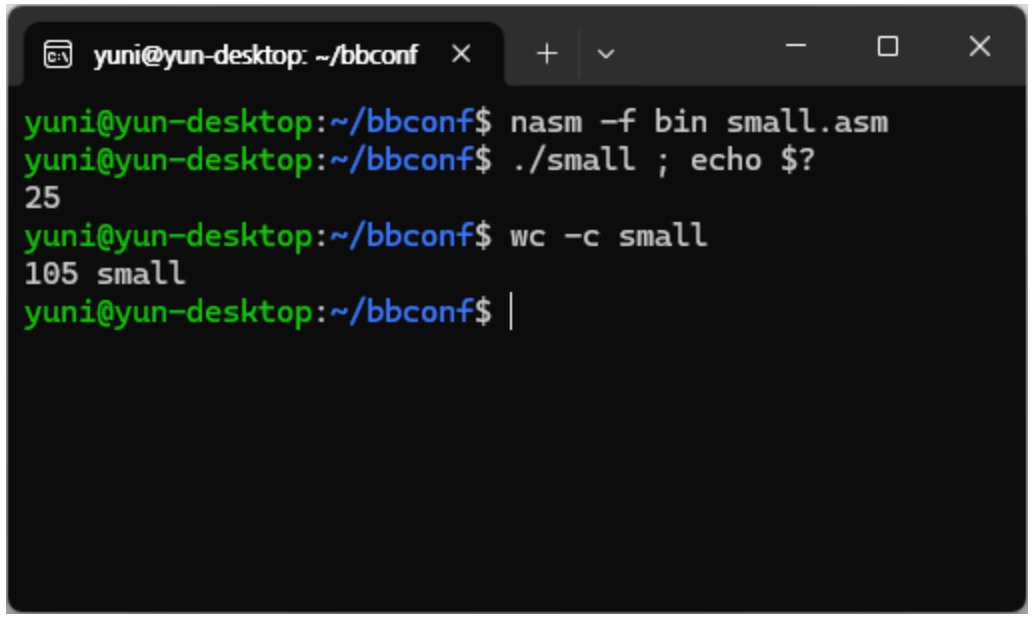
Best record **112** Bytes

활용 가능한 헤더의 필드들



Best record **112** Bytes

활용 가능한 헤더의 필드들



Best record **112** -> **105** Bytes (▼7)

활용 가능한 헤더의 필드들

이후 수많은 시행착오들...

Best record **105** Bytes

활용 가능한 헤더의 필드들

이후 수많은 시행착오들...

- base address는 0x10000 이상, 0x7ffffffe000 이하일 것

활용 가능한 헤더의 필드들

이후 수많은 시행착오들...

- base address는 0x10000 이상, 0x7ffffffe000 이하일 것
- p_flags는 0(executable), 2(readable)이 켜져있어야 하지만 0만 켜져있어도 가능

활용 가능한 헤더의 필드들

이후 수많은 시행착오들...

- base address는 0x10000 이상, 0x7ffffffe000 이하일 것
- p_flags는 0(executable), 2(readable)이 켜져있어야 하지만 0만 켜져있어도 가능
- p_filesz는 0 초과면 상관 없음

활용 가능한 헤더의 필드들

이후 수많은 시행착오들...

- base address는 0x10000 이상, 0x7ffffffe000 이하일 것
- p_flags는 0(executable), 2(readable)이 켜져있어야 하지만 0만 켜져있어도 가능
- p_filesz는 0 초과면 상관 없음
- p_memsz는 p_filesz와 동일하면 상관 없음

활용 가능한 헤더의 필드들

이후 수많은 시행착오들...

- base address는 0x10000 이상, 0x7ffffffe000 이하일 것
- p_flags는 0(executable), 2(readable)이 켜져있어야 하지만 0만 켜져있어도 가능
- p_filesz는 0 초과면 상관 없음
- p_memsz는 p_filesz와 동일하면 상관 없음
- p_offset이 x(단, x는 4096 미만)이면 p_vaddr는 org+x 값이어야 함 (???)

Best record **105** Bytes

활용 가능한 헤더의 필드들

이후 수많은 시행착오들...

- base address는 0x10000 이상, 0x7ffffffe000 이하일 것
- p_flags는 0(executable), 2(readable)이 켜져있어야 하지만 0만 켜져있어도 가능
- p_filesz는 0 초과면 상관 없음
- p_memsz는 p_filesz와 동일하면 상관 없음
- p_offset이 x(단, x는 4096 미만)이면 p_vaddr는 org+x 값이어야 함 (???)
- E, L, F는 머신 코드로 REX 프리픽스에 해당, e_entry는 org+1이어도 가능 (???)

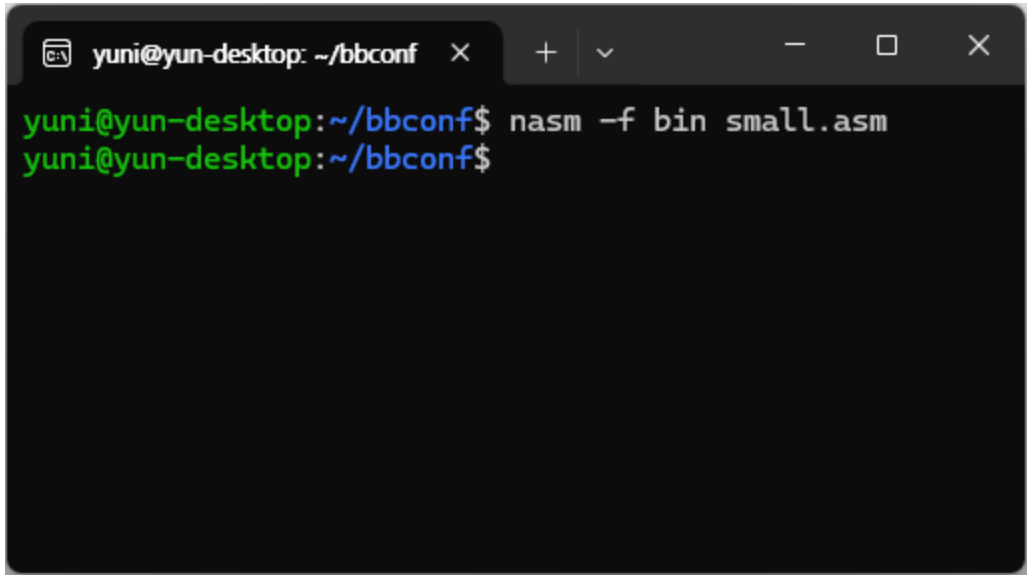
활용 가능한 헤더의 필드들

		e_entry	p_type	0x01
				0x00
				0x00
				0x00
		0x01	p_flags	홀수면 됨
	0x00			
	0x00			
	0x18	e_phoff	p_offset	0x18
	0x00			0x00
	0x00			0x00
	0x00			0x00
	0x00			0x00
	0x00			0x00
	0x00			0x00
		e_shoff	p_vaddr	0x18
			0x01	
				0x00
				0x00
		e_flags	p_paddr	
		e_ehsize		
	0x38	e_phentsize		
	0x00			
	0x01	e_phnum	p_filesz	0보다 클 것
	0x00			
		e_shentsize		
		e_shnum		
		e_shstrndx		
			p_memsz	p_filesz와 동일할 것

```
yuni@yun-desktop: ~/bbconf

BITS 64

                org     0x0100000000

_start:         db      0x7f, "ELF"     ; e_ident[0..3]
                mov     al, 60          ; e_ident[4..5]
                mov     dil, 25         ; e_ident[6..8]
                syscall                 ; e_ident[9..10]
                db      0, 0, 0, 0, 0   ; e_ident[11..15]
                dw      2               ; e_type
                dw      0x3e            ; e_machine
                dd      0               ; e_version
                dd      1               ; e_entry       ; p_type
                dd      1                               ; p_flags
                dq      0x18            ; e_phoff       ; p_offset
                dd      0x18            ; e_shoff       ; p_vaddr
                dd      1
                dd      0               ; e_flags       ; p_paddr
                dw      0               ; e_ehsize
                dw      0x38            ; e_phentsize
                dw      1               ; e_phnum       ; p_filesz
                dw      0               ; e_shentsize
                dw      0               ; e_shnum
                dw      0               ; e_shstrndx
                dq      1                               ; p_memsz
                dq      0                               ; p_align
~
~
                                                27,0-1        All
```

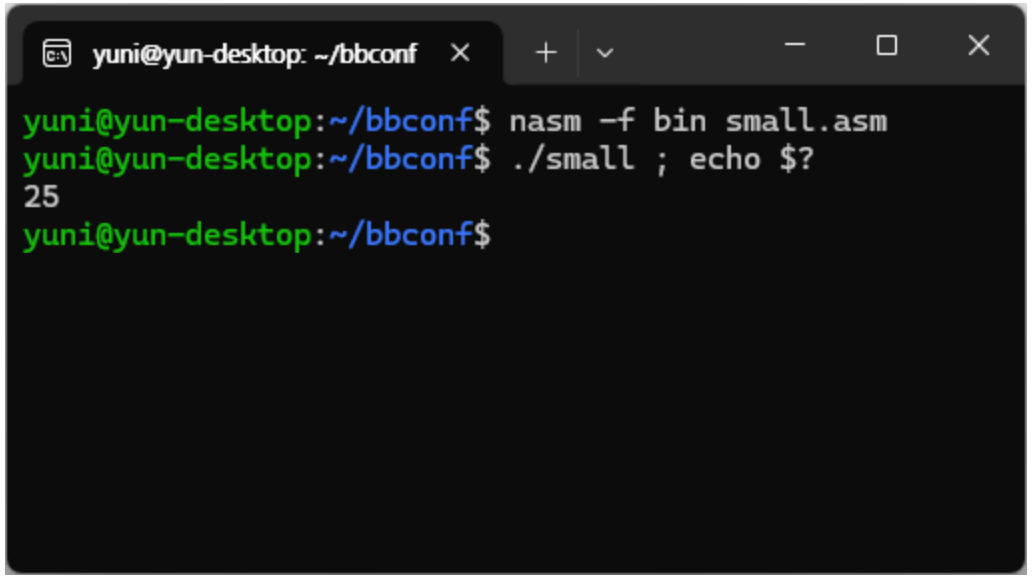
Best record **105** Bytes

활용 가능한 헤더의 필드들



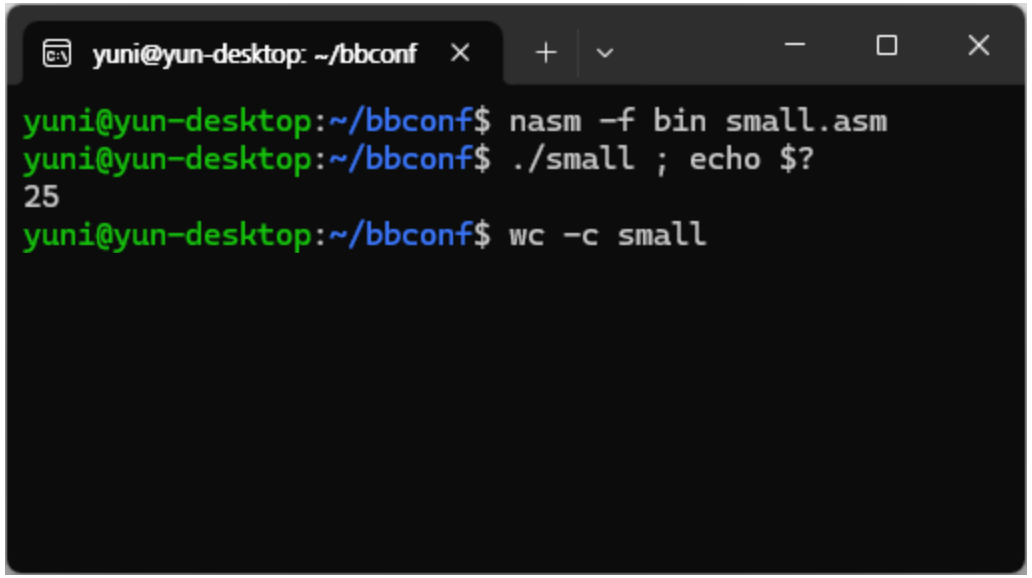
Best record **105** Bytes

활용 가능한 헤더의 필드들



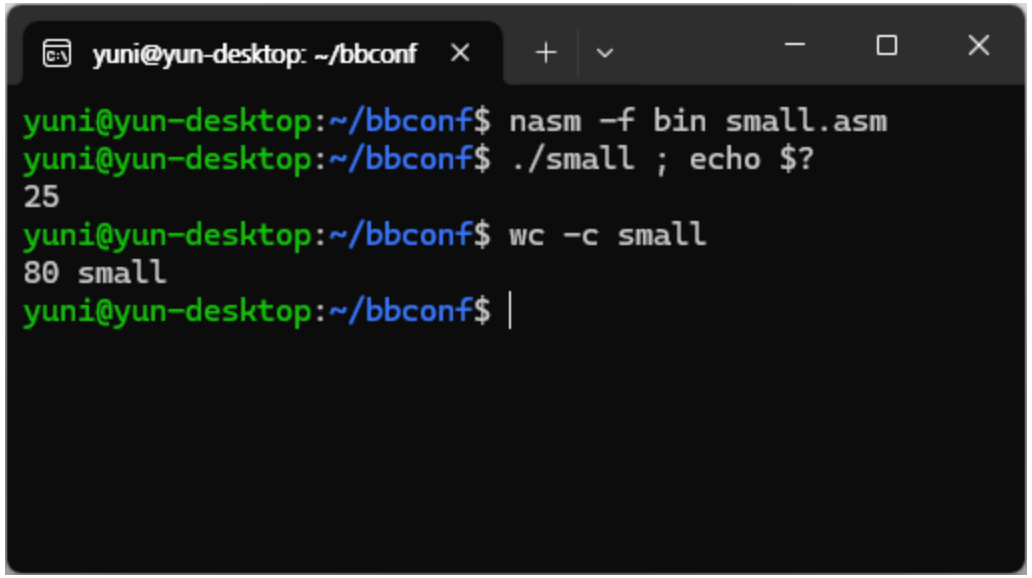
Best record **105** Bytes

활용 가능한 헤더의 필드들



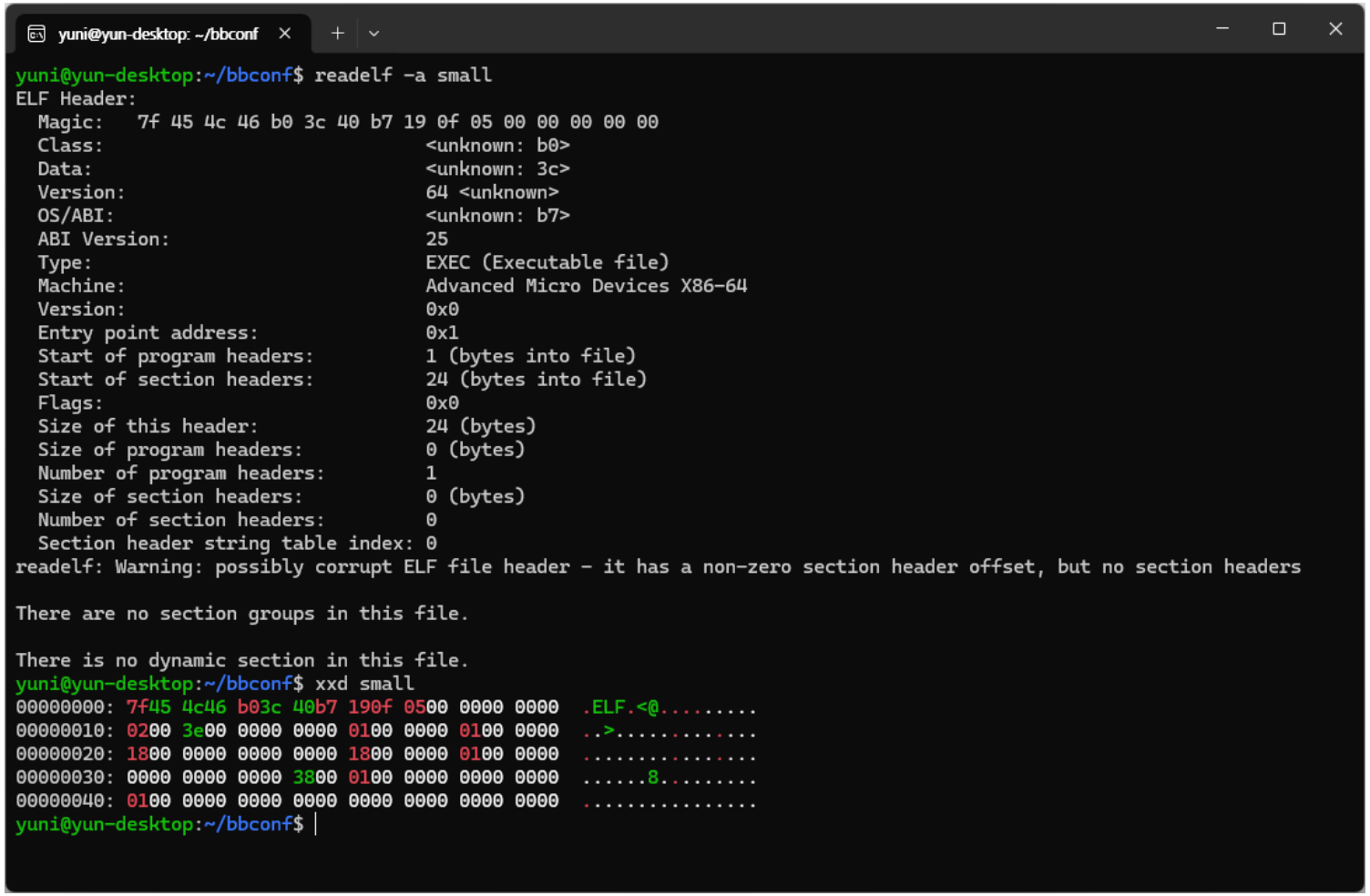
Best record **105** Bytes

활용 가능한 헤더의 필드들



Best record **105** -> **80** Bytes (▼25)

최종 결과물



Best record **80** Bytes

이제 헬로월드를 출력해야 한다

Best record **80** Bytes

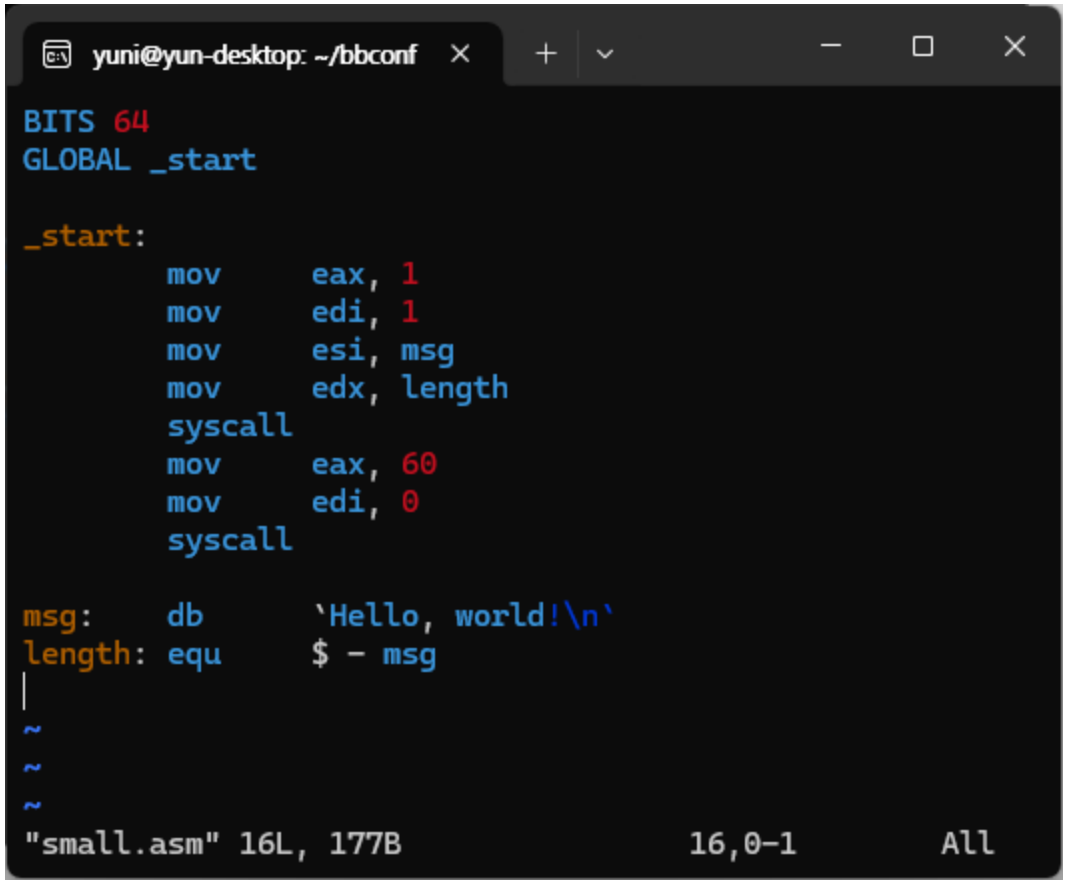
이제 헬로월드를 출력해야 한다

```
yuni@yun-desktop:~/bbconf$ xxd layout
00000000: 7f45 4c46 cccc cccc cccc cccc cccc cccc  .ELF............
00000010: 0200 3e00 cccc cccc 0100 0000 ddee 0000  ..>.............
00000020: 1800 0000 0000 0000 1800 0000 ddee 0000  ................
00000030: cccc cccc cccc 3800 0100 ffff ff00 0000  ......8.........
00000040: 0100 ffff ff00 0000 cccc cccc cccc cccc  ................
```

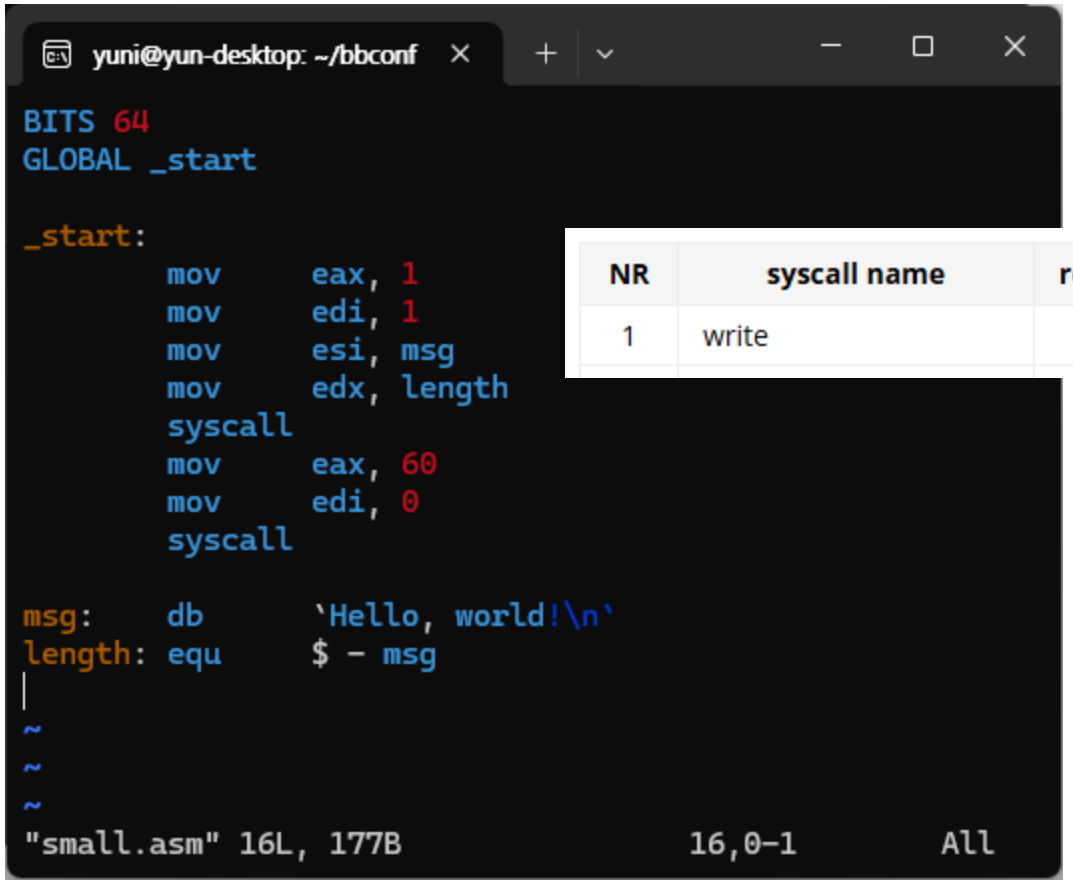
0xcc : 자유롭게 쓸 수 있는 공간

그 외에는 1쌍씩 값이 서로 같아야 함
0xdd : 홀수여야 함
0xee : 0x7f 이하여야 함
0xff : 3바이트 값이 0x200000 미만이면 가끔 성공 (?)

이제 헬로월드를 출력해야 한다

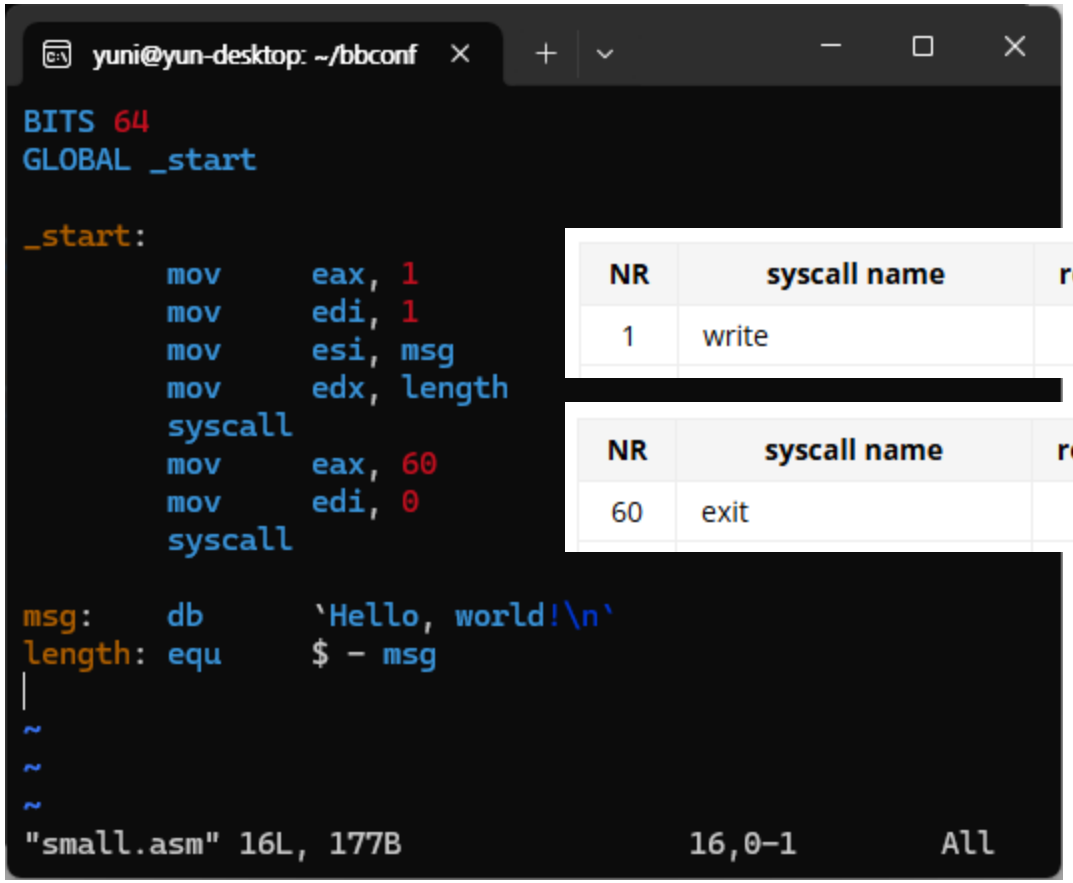


이제 헬로월드를 출력해야 한다



NR	syscall name	references	%rax	arg0 (%rdi)	arg1 (%rsi)	arg2 (%rdx)
1	write	man/ cs/	0x01	unsigned int fd	const char *buf	size_t count

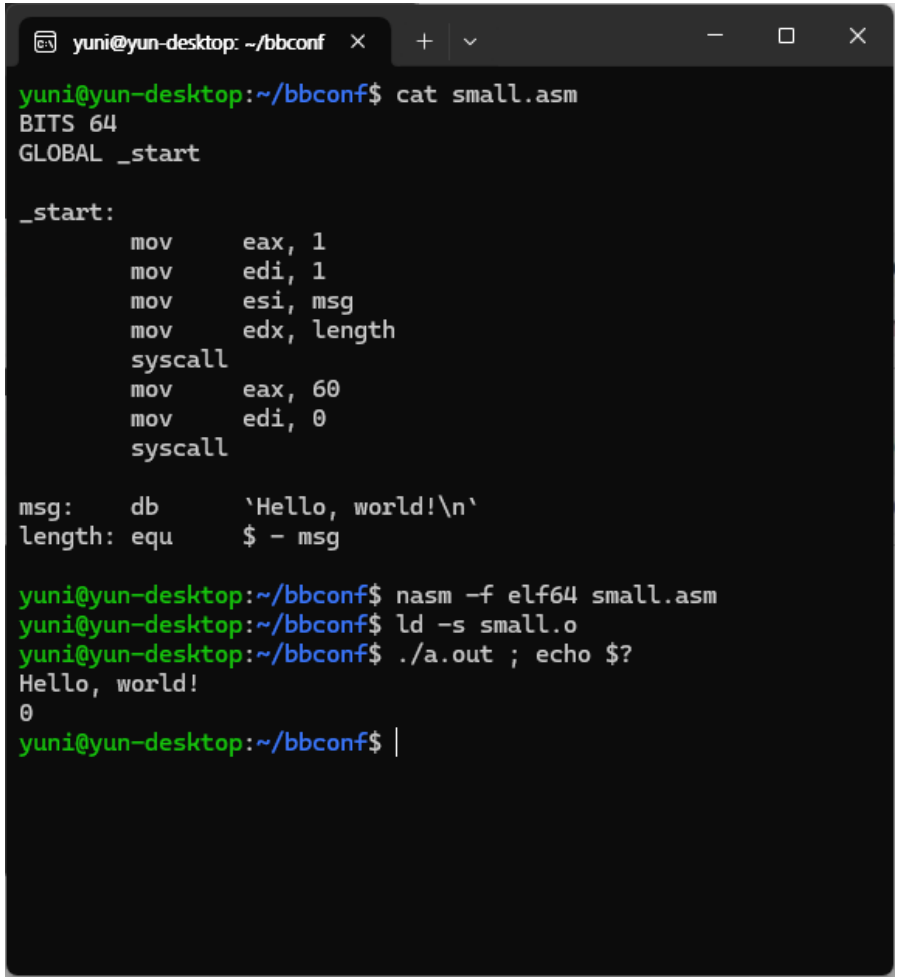
이제 헬로월드를 출력해야 한다



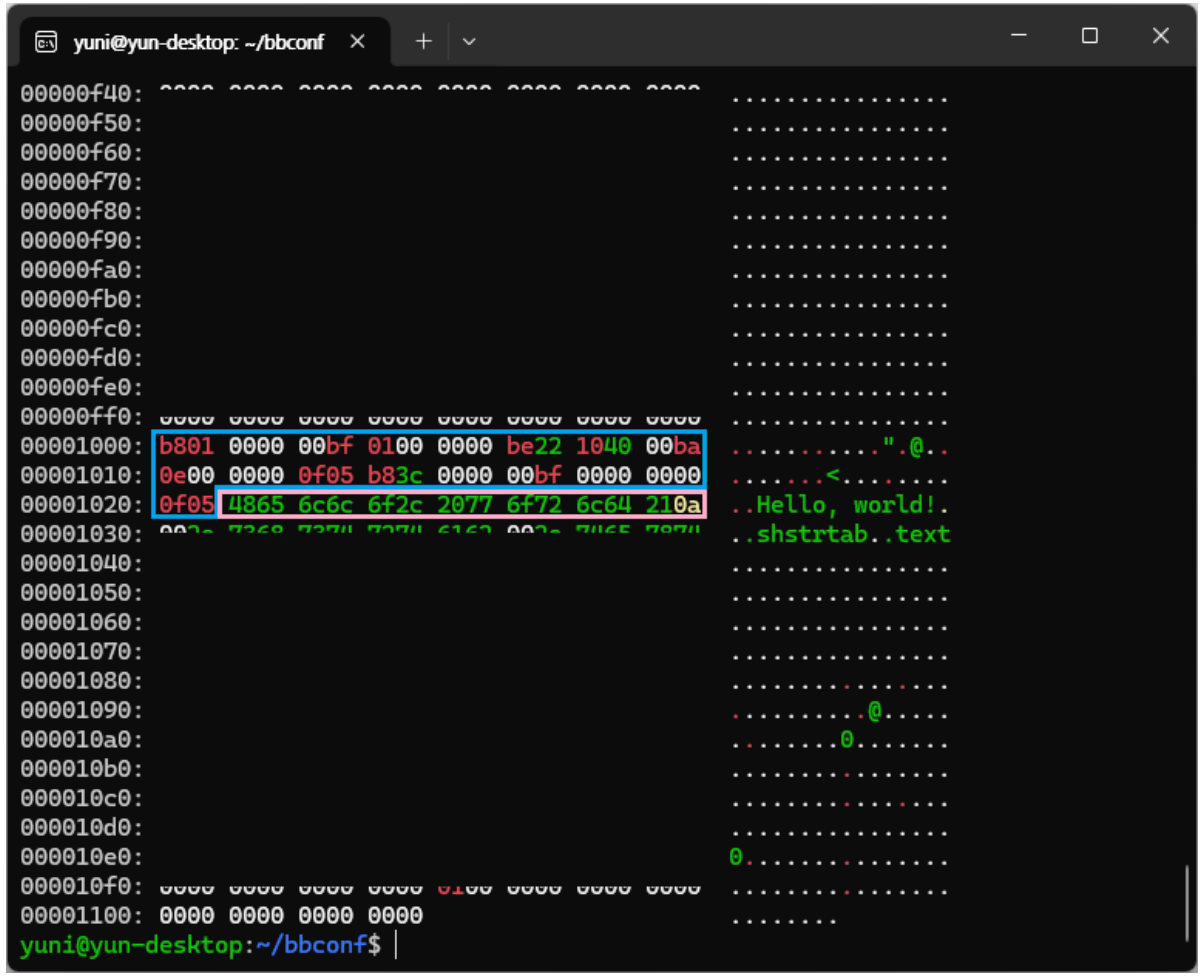
NR	syscall name	references	%rax	arg0 (%rdi)	arg1 (%rsi)	arg2 (%rdx)
1	write	man/ cs/	0x01	unsigned int fd	const char *buf	size_t count

NR	syscall name	references	%rax	arg0 (%rdi)
60	exit	man/ cs/	0x3c	int error_code

이제 헬로월드를 출력해야 한다

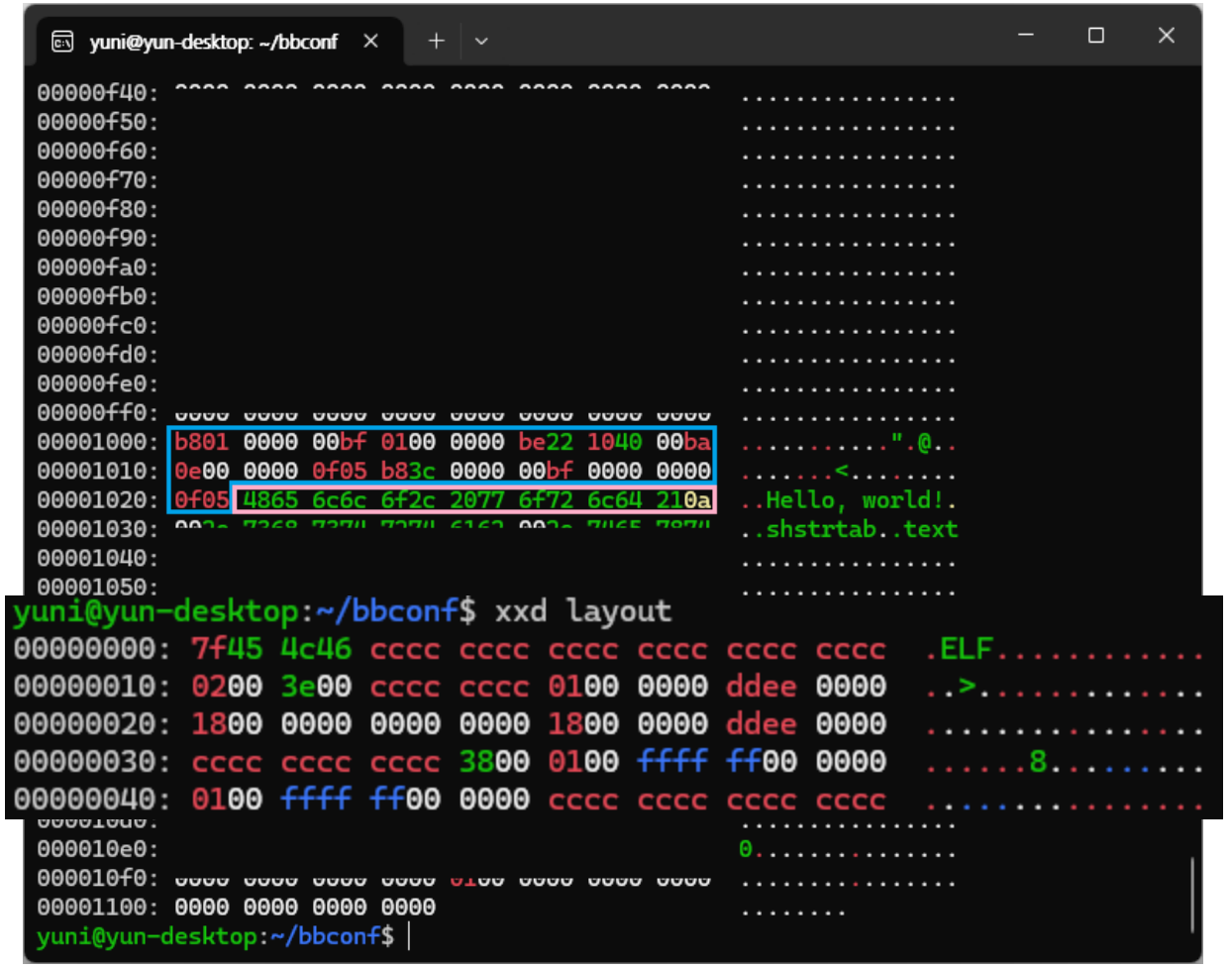


이제 헬로월드를 출력해야 한다



명령어 : 34 bytes
데이터 : 14 bytes

이제 헬로월드를 출력해야 한다



명령어 : 34 bytes
데이터 : 14 bytes

이제 헬로월드를 출력해야 한다

명령어 중간중간에 jmp를 넣어야 함

jmp 명령어는 eb <offset> 꼴로, 2바이트 소모

적당히 넣어보기

Disassembly:

```
0:  b8 01 00 00 00          mov    eax,0x1
5:  bf 01 00 00 00          mov    edi,0x1
a:  48 be ff 00 00 00 01    movabs rsi,0x1000000ff
11: 00 00 00
14: ba 0e 00 00 00          mov    edx,0xe
19: 0f 05                   syscall
1b: b8 3c 00 00 00          mov    eax,0x3c
20: bf 00 00 00 00          mov    edi,0x0
25: 0f 05                   syscall
```

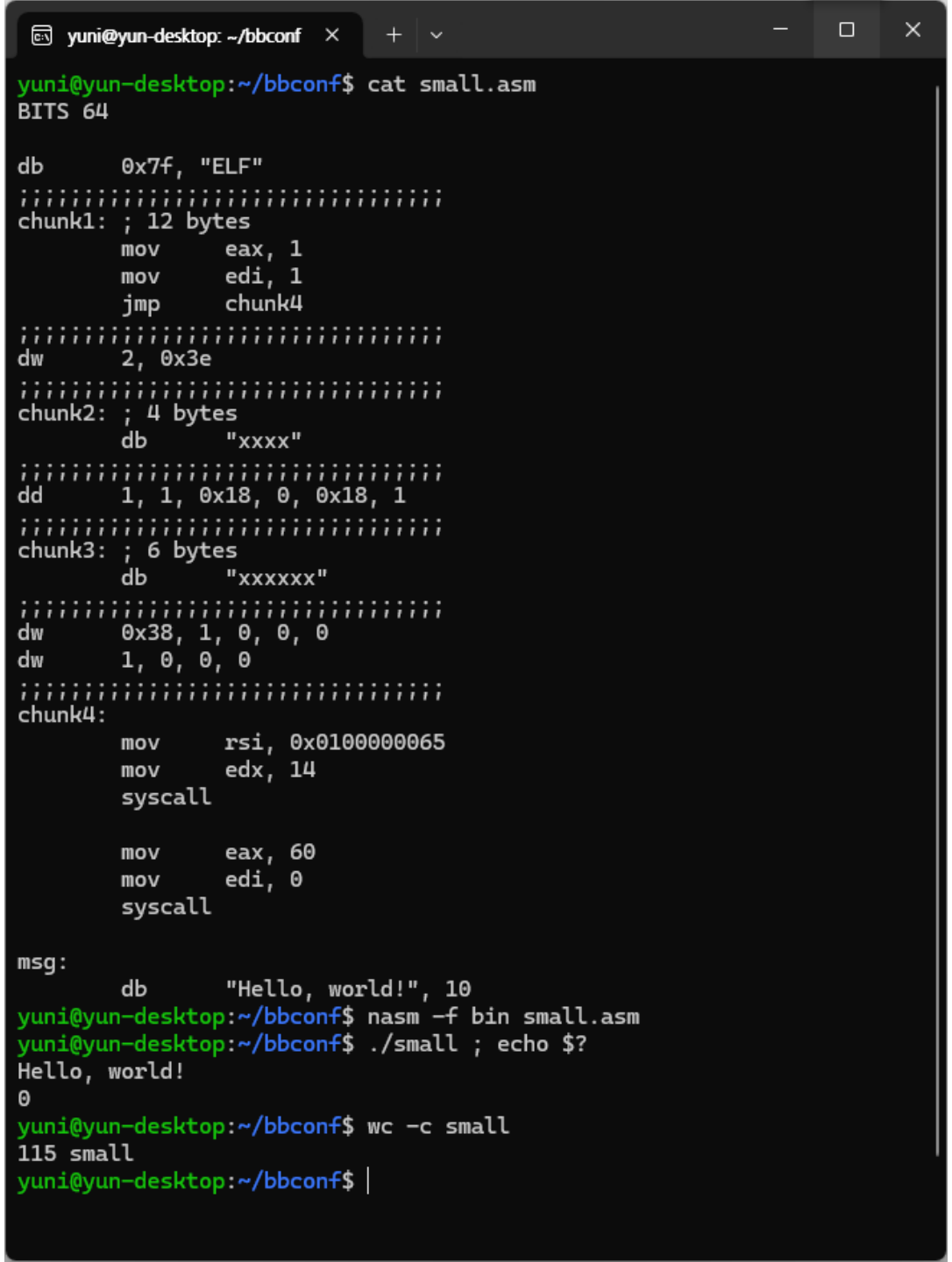
적당히 넣어보기

Disassembly:

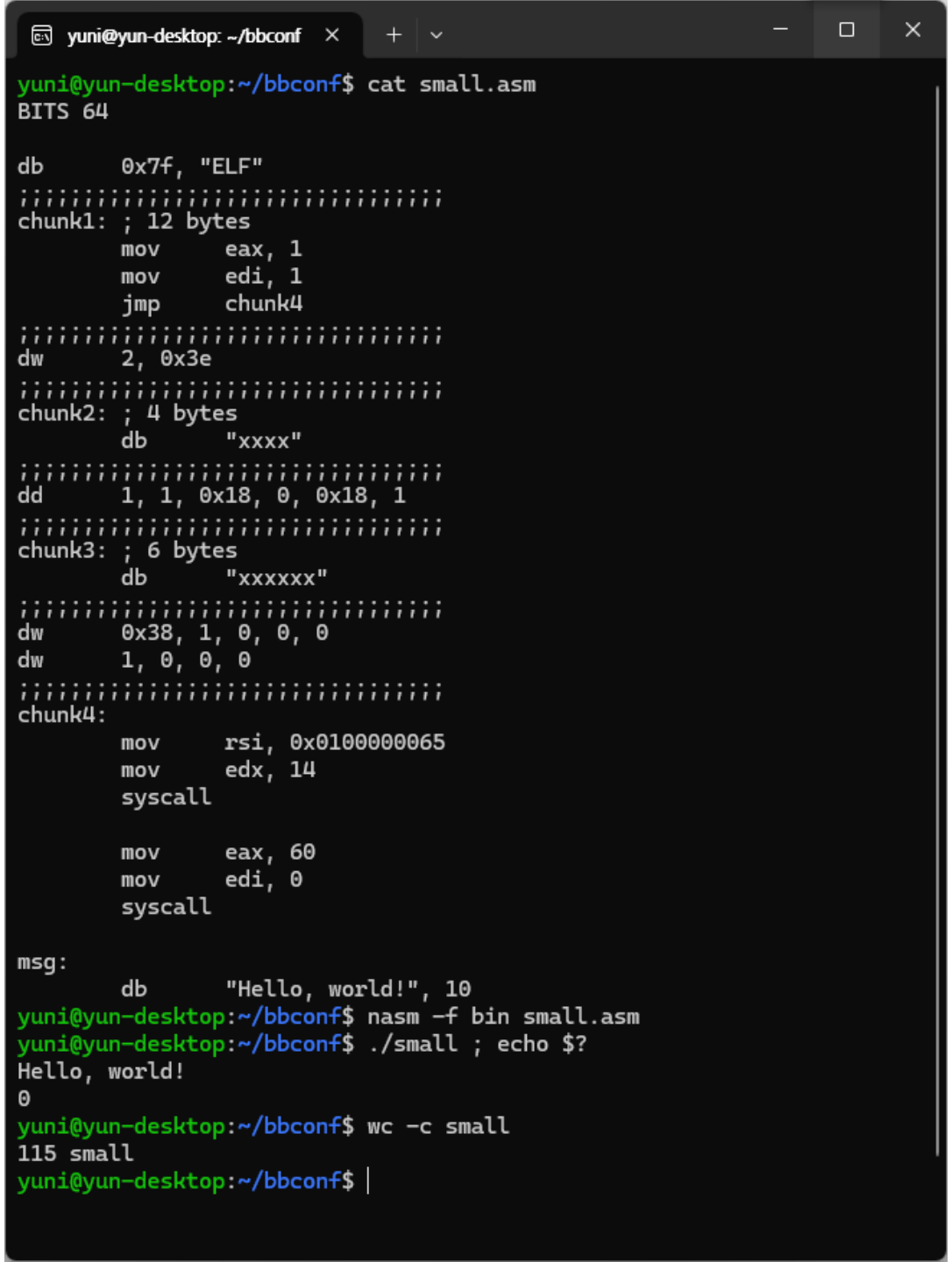
```
0:   b8 01 00 00 00          mov    eax,0x1
5:   bf 01 00 00 00          mov    edi,0x1
a:   48 be ff 00 00 00 01    movabs rsi,0x1000000ff
11:  00 00 00
14:  ba 0e 00 00 00          mov    edx,0xe
19:  0f 05                   syscall
1b:  b8 3c 00 00 00          mov    eax,0x3c
20:  bf 00 00 00 00          mov    edi,0x0
25:  0f 05                   syscall
```

jmp

적당히 넣어보기


yuni@yun-desktop: ~/bbconf
yuni@yun-desktop:~/bbconf$ cat small.asm
BITS 64
db 0x7f, "ELF"
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
chunk1: ; 12 bytes
mov eax, 1
mov edi, 1
jmp chunk4
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
dw 2, 0x3e
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
chunk2: ; 4 bytes
db "xxxx"
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
dd 1, 1, 0x18, 0, 0x18, 1
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
chunk3: ; 6 bytes
db "xxxxxx"
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
dw 0x38, 1, 0, 0, 0
dw 1, 0, 0, 0
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
chunk4:
mov rsi, 0x0100000065
mov edx, 14
syscall
mov eax, 60
mov edi, 0
syscall
msg:
db "Hello, world!", 10
yuni@yun-desktop:~/bbconf$ nasm -f bin small.asm
yuni@yun-desktop:~/bbconf$ ./small ; echo $?
Hello, world!
0
yuni@yun-desktop:~/bbconf$ wc -c small
115 small
yuni@yun-desktop:~/bbconf$

적당히 넣어보기



Best record **115** Bytes

어셈블리 컷팅

Disassembly:

```
0:  b8 01 00 00 00          mov    eax,0x1
5:  bf 01 00 00 00          mov    edi,0x1
a:  48 be ff 00 00 00 01    movabs rsi,0x1000000ff
11: 00 00 00
14: ba 0e 00 00 00          mov    edx,0xe
19: 0f 05                   syscall
1b: b8 3c 00 00 00          mov    eax,0x3c
20: bf 00 00 00 00          mov    edi,0x0
25: 0f 05                   syscall
```

Best record **115** Bytes

어셈블리 컷팅

Disassembly:

```
0:   b0 01                   mov    al,0x1
2:   40 b7 01                mov    dil,0x1
5:   48 be ff 00 00 00 01    movabs rsi,0x1000000ff
c:   00 00 00
f:   b2 0e                   mov    dl,0xe
11:  0f 05                   syscall
13:  b0 3c                   mov    al,0x3c
15:  40 b7 00                mov    dil,0x0
18:  0f 05                   syscall
```

어차피 레지스터들은 초기에 0이므로,
하위 1바이트 값만 바꿔도 충분

Best record **115** Bytes

어셈블리 컷팅

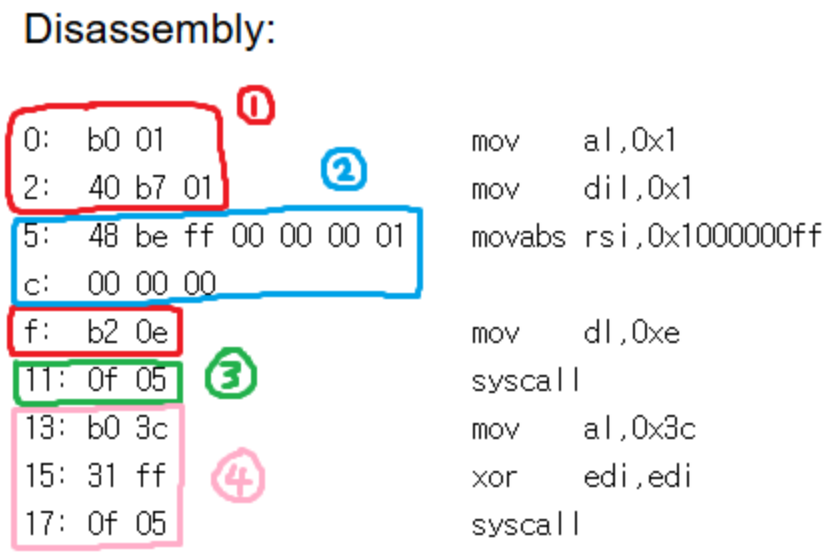
Disassembly:

```
0:  b0 01                   mov    al,0x1
2:  40 b7 01                mov    dil,0x1
5:  48 be ff 00 00 00 01    movabs rsi,0x1000000ff
c:  00 00 00
f:  b2 0e                   mov    dl,0xe
11: 0f 05                   syscall
13: b0 3c                   mov    al,0x3c
15: 31 ff                   xor    edi,edi
17: 0f 05                   syscall
```

rdi를 0으로 만들 때는
mov보다 xor이 1바이트 짧다

Best record **115** Bytes

어셈블리 컷팅



뒷부분을 욱여넣을 수 있음

Best record **115** Bytes

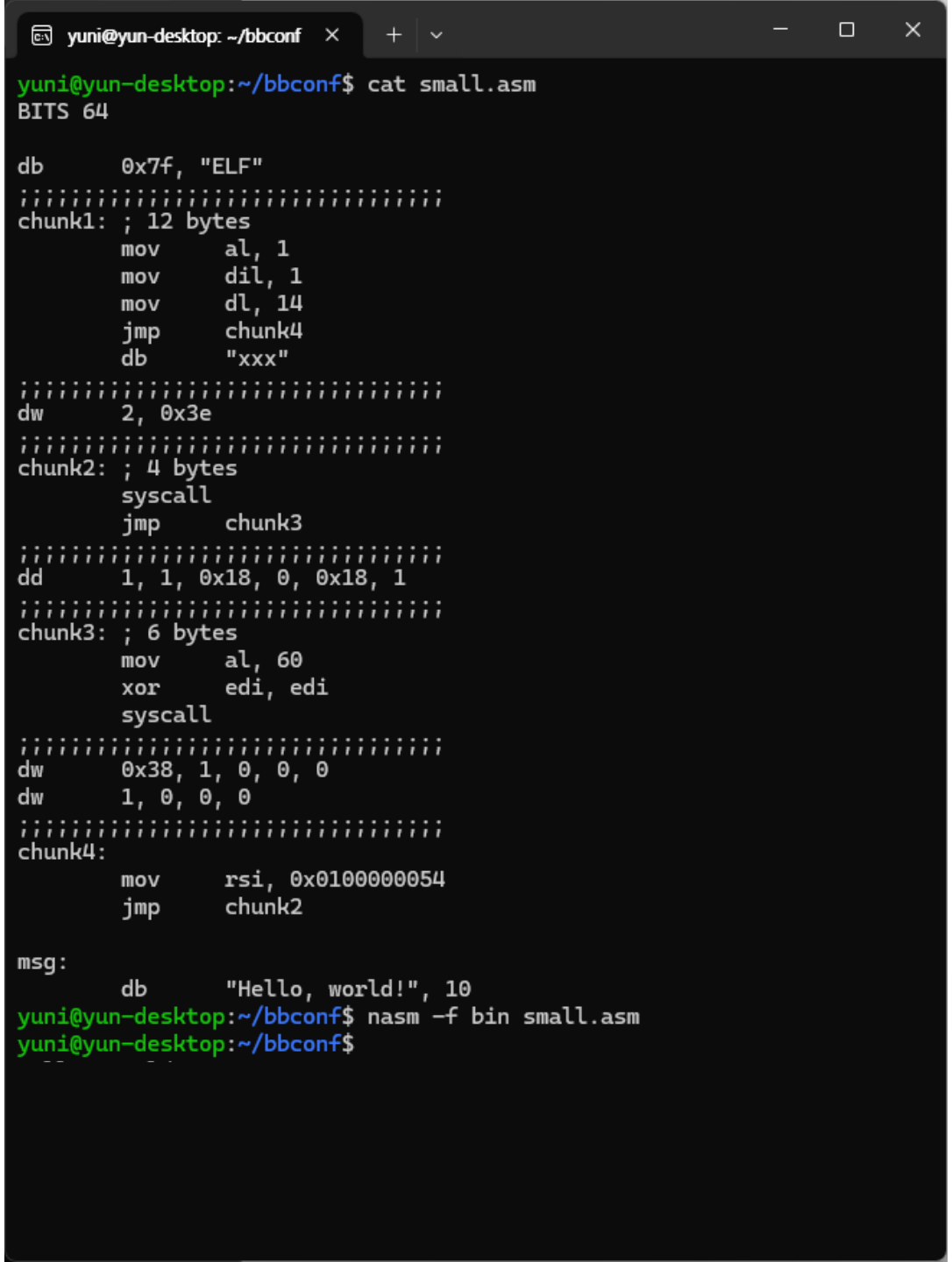
어셈블리 컷팅

yuni@yun-desktop: ~/bbconf

```
yuni@yun-desktop:~/bbconf$ cat small.asm
BITS 64

db      0x7f, "ELF"
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
chunk1: ; 12 bytes
        mov     al, 1
        mov     dil, 1
        mov     dl, 14
        jmp     chunk4
        db      "xxx"
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
dw      2, 0x3e
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
chunk2: ; 4 bytes
        syscall
        jmp     chunk3
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
dd      1, 1, 0x18, 0, 0x18, 1
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
chunk3: ; 6 bytes
        mov     al, 60
        xor     edi, edi
        syscall
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
dw      0x38, 1, 0, 0, 0
dw      1, 0, 0, 0
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
chunk4:
        mov     rsi, 0x0100000054
        jmp     chunk2

msg:
        db      "Hello, world!", 10
yuni@yun-desktop:~/bbconf$
```

Best record **115** Bytes

어셈블리 컷팅



Best record **115** Bytes

어셈블리 컷팅

yuni@yun-desktop: ~/bbconf

```
yuni@yun-desktop:~/bbconf$ cat small.asm
BITS 64

db      0x7f, "ELF"
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
chunk1: ; 12 bytes
        mov     al, 1
        mov     dil, 1
        mov     dl, 14
        jmp     chunk4
        db      "xxx"
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
dw      2, 0x3e
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
chunk2: ; 4 bytes
        syscall
        jmp     chunk3
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
dd      1, 1, 0x18, 0, 0x18, 1
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
chunk3: ; 6 bytes
        mov     al, 60
        xor     edi, edi
        syscall
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
dw      0x38, 1, 0, 0, 0
dw      1, 0, 0, 0
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
chunk4:
        mov     rsi, 0x0100000054
        jmp     chunk2

msg:
        db      "Hello, world!", 10
yuni@yun-desktop:~/bbconf$ nasm -f bin small.asm
yuni@yun-desktop:~/bbconf$ ./small ; echo $?
Hello, world!
0
yuni@yun-desktop:~/bbconf$
```

Best record **115** Bytes

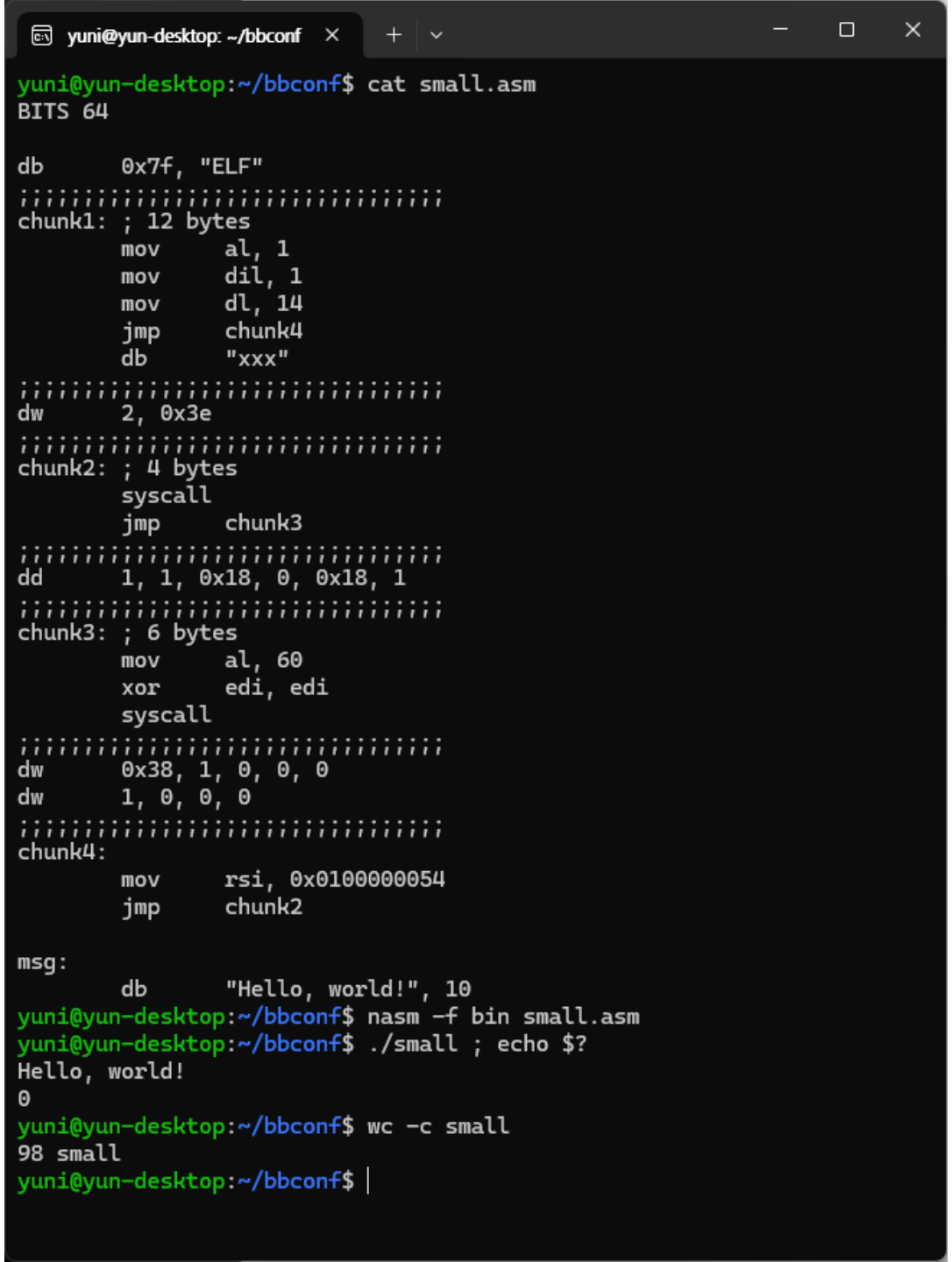
어셈블리 컷팅

yuni@yun-desktop: ~/bbconf

```
yuni@yun-desktop:~/bbconf$ cat small.asm
BITS 64

db      0x7f, "ELF"
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
chunk1: ; 12 bytes
        mov     al, 1
        mov     dil, 1
        mov     dl, 14
        jmp     chunk4
        db      "xxx"
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
dw      2, 0x3e
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
chunk2: ; 4 bytes
        syscall
        jmp     chunk3
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
dd      1, 1, 0x18, 0, 0x18, 1
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
chunk3: ; 6 bytes
        mov     al, 60
        xor     edi, edi
        syscall
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
dw      0x38, 1, 0, 0, 0
dw      1, 0, 0, 0
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
chunk4:
        mov     rsi, 0x0100000054
        jmp     chunk2

msg:
        db      "Hello, world!", 10
yuni@yun-desktop:~/bbconf$ nasm -f bin small.asm
yuni@yun-desktop:~/bbconf$ ./small ; echo $?
Hello, world!
0
yuni@yun-desktop:~/bbconf$ wc -c small
```

Best record **115** Bytes

어셈블리 컷팅



Best record **115** -> **98** Bytes (▼17)

8바이트 주솟값 우회

Disassembly:

```
0:   b0 01                   mov    al,0x1
2:   40 b7 01                mov    dil,0x1
5:   b2 0e                   mov    dl,0xe
7:   48 be ff 00 00 00 01    movabs rsi,0x1000000ff
e:   00 00 00
11:  0f 05                   syscall
13:  b0 3c                   mov    al,0x3c
15:  31 ff                   xor    edi,edi
17:  0f 05                   syscall
```

솔직히 인간적으로
혼자 10바이트 차지하는 게 괘씸함

Best record **98** Bytes

8바이트 주솟값 우회

Disassembly:

```
0:   b0 01                  mov    al,0x1
2:   40 b7 01               mov    dil,0x1
5:   b2 0e                  mov    dl,0xe
7:   ff c6                  inc    esi
9:   48 c1 e6 20            shl    rsi,0x20
d:   40 b6 ff               mov    sil,0xff
10:  0f 05                  syscall
12:  b0 3c                  mov    al,0x3c
14:  31 ff                  xor    edi,edi
16:  0f 05                  syscall
```

소중한 1바이트 절약

Best record **98** Bytes

8바이트 주솟값 우회

```
yuni@yun-desktop:~/bbconf$ cat small.asm
BITS 64

db      0x7f, "ELF"
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
chunk1: ; 12 bytes
        mov     al, 1
        mov     dil, 1
        mov     dl, 14
        jmp     chunk4
        db      "xxx"
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
dw      2, 0x3e
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
chunk2: ; 4 bytes
        syscall
        jmp     chunk3
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
dd      1, 1, 0x18, 0, 0x18, 1
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
chunk3: ; 6 bytes
        mov     al, 60
        xor     edi, edi
        syscall
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
dw      0x38, 1, 0, 0, 0
dw      1, 0, 0, 0
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
chunk4:
        inc     esi
        shl     rsi, 32
        mov     sil, 0x53
        jmp     chunk2

msg:
        db      "Hello, world!", 10
yuni@yun-desktop:~/bbconf$ nasm -f bin small.asm
yuni@yun-desktop:~/bbconf$ ./small ; echo $?
Hello, world!
0
yuni@yun-desktop:~/bbconf$ wc -c small
97 small
yuni@yun-desktop:~/bbconf$
```

Best record **98** -> **97** Bytes (▼1)

주소를 만들지 말고 얻어낼 것

Disassembly:

```
0:   b0 01               mov    al,0x1
2:   40 b7 01            mov    dil,0x1
5:   b2 0e               mov    dl,0xe
7:   ff c6               inc    esi
9:   48 c1 e6 20         shl    rsi,0x20
d:   40 b6 ff            mov    sil,0xff
10:  0f 05               syscall
12:  b0 3c               mov    al,0x3c
14:  31 ff               xor    edi,edi
16:  0f 05               syscall
```

Best record **97** Bytes

주소를 만들지 말고 얻어낼 것

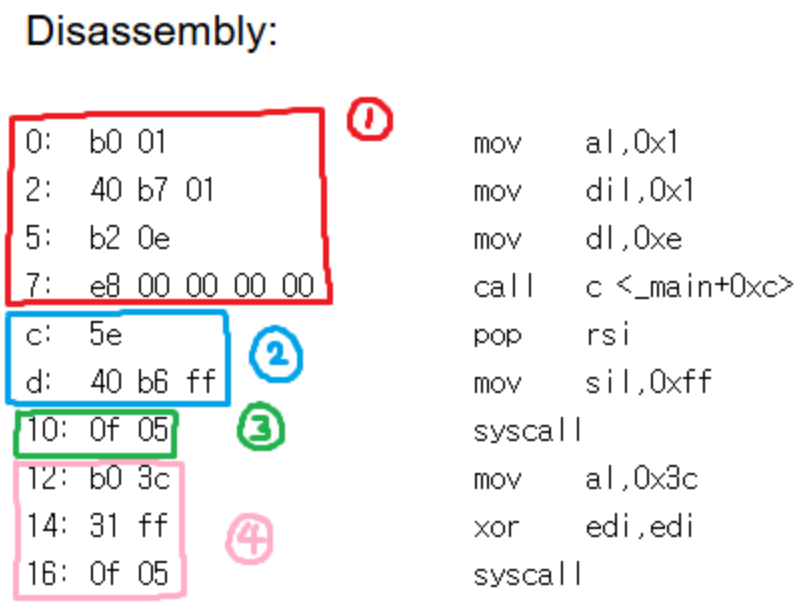
Disassembly:

```
0:   b0 01                  mov    al,0x1
2:   40 b7 01               mov    dil,0x1
5:   b2 0e                  mov    dl,0xe
7:   e8 00 00 00 00         call   c <_main+0xc>
c:   5e                     pop    rsi
d:   40 b6 ff               mov    sil,0xff
10:  0f 05                  syscall
12:  b0 3c                  mov    al,0x3c
14:  31 ff                  xor    edi,edi
16:  0f 05                  syscall
```

call = push + jmp
멋진 테크닉이지만 길이는 동일

Best record **97** Bytes

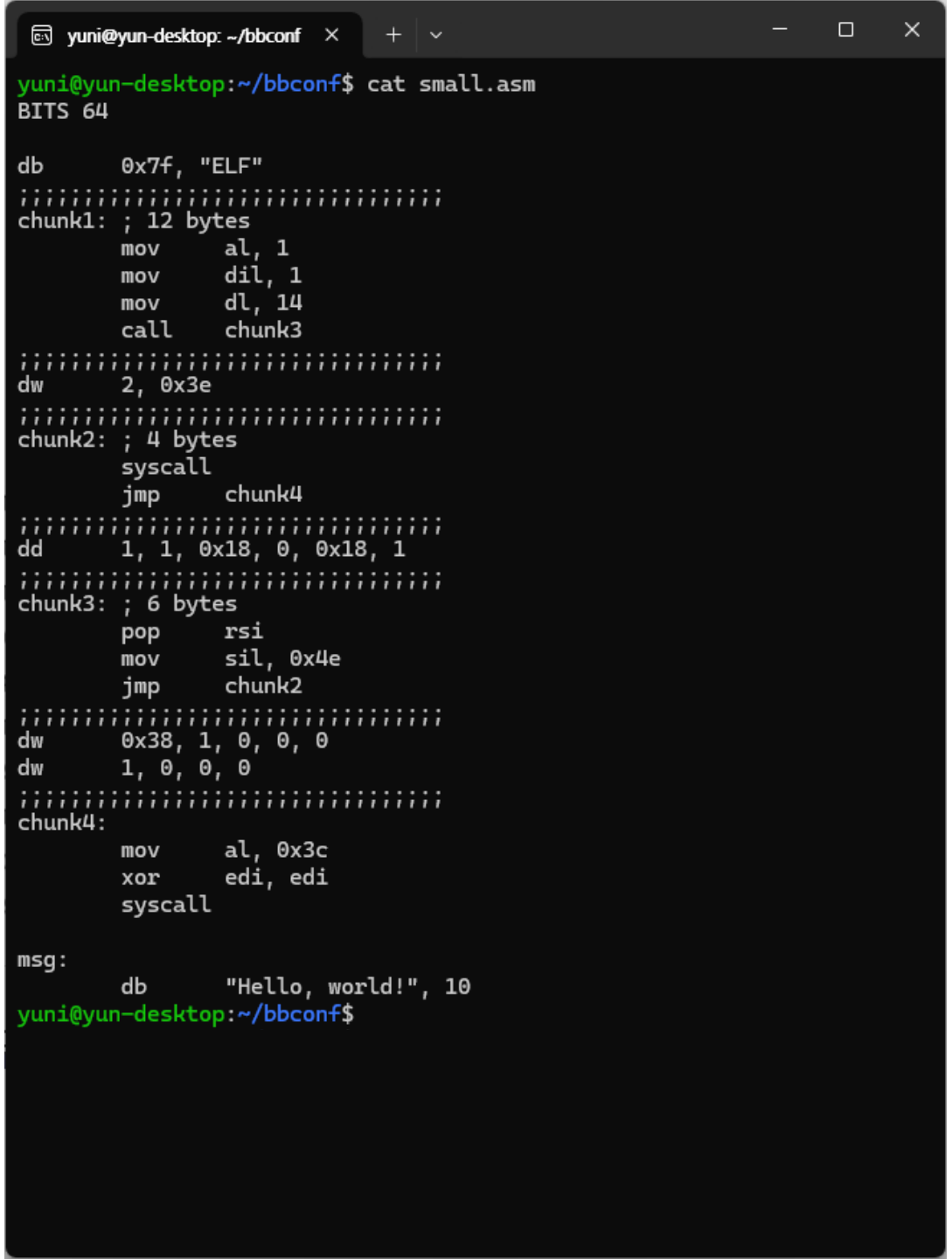
주소를 만들지 말고 얻어낼 것



하지만 넣을 수 있는 위치에 변화가 생김

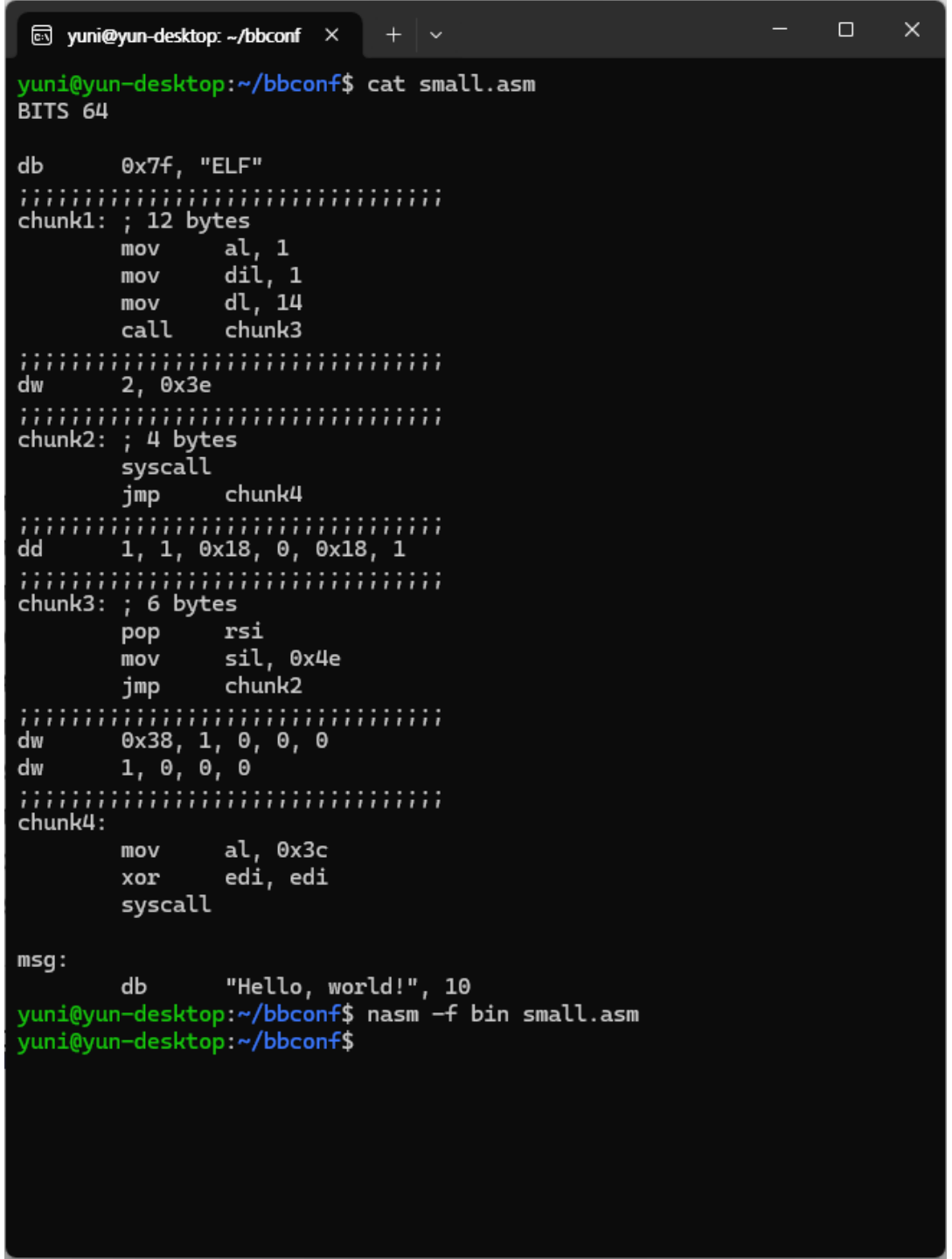
Best record **97** Bytes

주소를 만들지 말고 얻어낼 것



Best record **97** Bytes

주소를 만들지 말고 얻어낼 것



Best record **97** Bytes

주소를 만들지 말고 얻어낼 것

```
yuni@yun-desktop:~/bbconf$ cat small.asm
BITS 64

db      0x7f, "ELF"
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
chunk1: ; 12 bytes
        mov     al, 1
        mov     dil, 1
        mov     dl, 14
        call    chunk3
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
dw      2, 0x3e
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
chunk2: ; 4 bytes
        syscall
        jmp     chunk4
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
dd      1, 1, 0x18, 0, 0x18, 1
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
chunk3: ; 6 bytes
        pop     rsi
        mov     sil, 0x4e
        jmp     chunk2
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
dw      0x38, 1, 0, 0, 0
dw      1, 0, 0, 0
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
chunk4:
        mov     al, 0x3c
        xor     edi, edi
        syscall

msg:
        db      "Hello, world!", 10
yuni@yun-desktop:~/bbconf$ nasm -f bin small.asm
yuni@yun-desktop:~/bbconf$ ./small ; echo $?
Hello, world!
0
yuni@yun-desktop:~/bbconf$
```

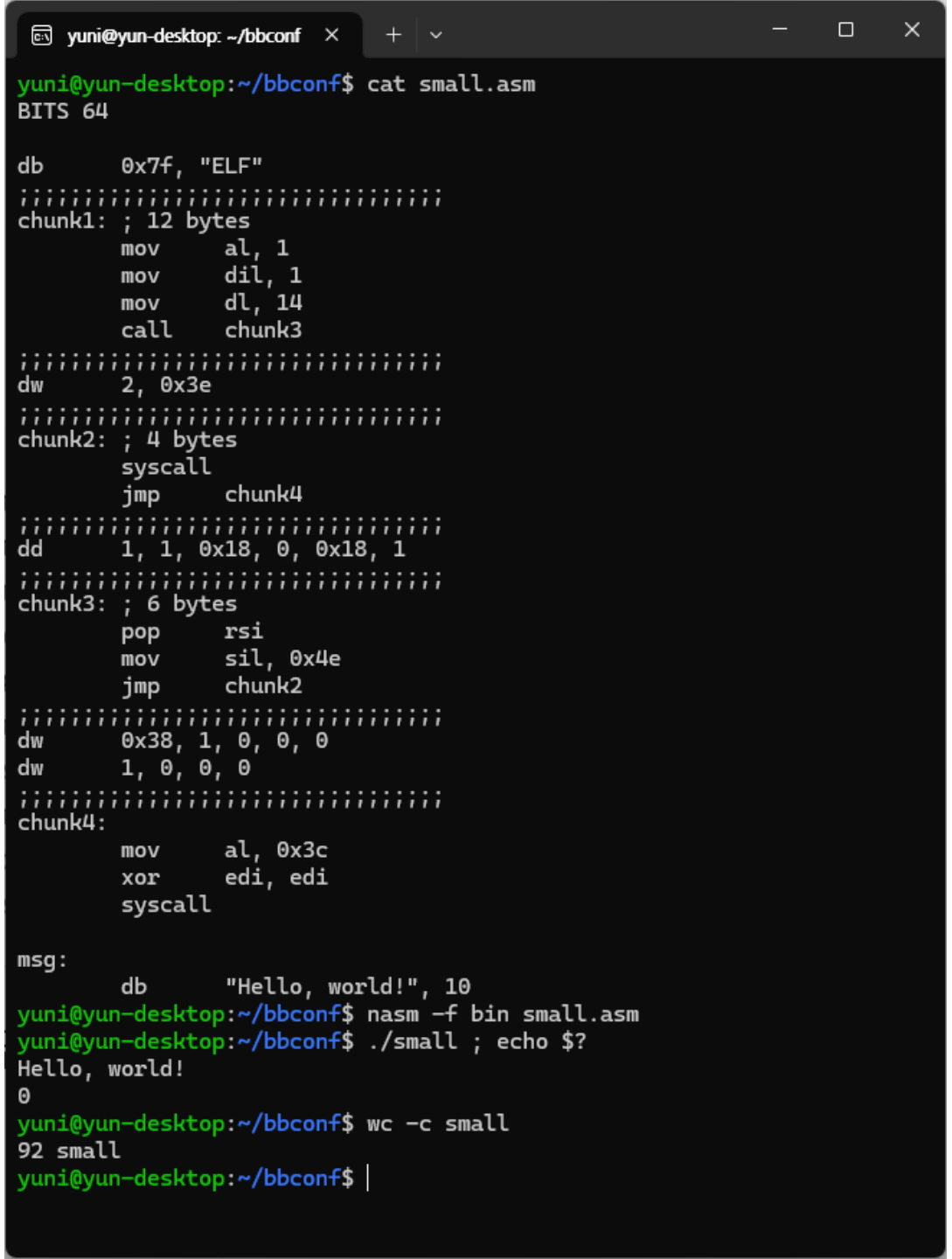
Best record **97** Bytes

주소를 만들지 말고 얻어낼 것

```
yuni@yun-desktop:~/bbconf$ cat small.asm
BITS 64

db      0x7f, "ELF"
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
chunk1: ; 12 bytes
        mov     al, 1
        mov     dil, 1
        mov     dl, 14
        call    chunk3
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
dw      2, 0x3e
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
chunk2: ; 4 bytes
        syscall
        jmp     chunk4
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
dd      1, 1, 0x18, 0, 0x18, 1
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
chunk3: ; 6 bytes
        pop     rsi
        mov     sil, 0x4e
        jmp     chunk2
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
dw      0x38, 1, 0, 0, 0
dw      1, 0, 0, 0
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
chunk4:
        mov     al, 0x3c
        xor     edi, edi
        syscall

msg:
        db      "Hello, world!", 10
yuni@yun-desktop:~/bbconf$ nasm -f bin small.asm
yuni@yun-desktop:~/bbconf$ ./small ; echo $?
Hello, world!
0
yuni@yun-desktop:~/bbconf$ wc -c small
```

Best record **97** Bytes

주소를 만들지 말고 얻어낼 것



Best record **97** -> **92** Bytes (▼5)

1바이트 명령어 활용

Disassembly:

```
0:   b0 01                  mov    al,0x1
2:   40 b7 01               mov    dil,0x1
5:   b2 0e                  mov    dl,0xe
7:   e8 00 00 00 00         call   c <_main+0xc>
c:   5e                     pop    rsi
d:   40 b6 ff               mov    sil,0xff
10:  0f 05                  syscall
12:  b0 3c                  mov    al,0x3c
14:  31 ff                  xor    edi,edi
16:  0f 05                  syscall
```

dil에 1 넣는 대신
edi에 eax 넣는 식으로 1바이트 절약

Best record **92** Bytes

1바이트 명령어 활용

Disassembly:

```
0:   b0 01                  mov    al,0x1
2:   89 c7                  mov    edi,eax
4:   b2 0e                  mov    dl,0xe
6:   e8 00 00 00 00         call   b <_main+0xb>
b:   5e                     pop    rsi
c:   40 b6 ff               mov    sil,0xff
f:   0f 05                  syscall
11:  b0 3c                  mov    al,0x3c
13:  31 ff                  xor    edi,edi
15:  0f 05                  syscall
```

dil에 1 넣는 대신
edi에 eax 넣는 식으로 1바이트 절약

Best record **92** Bytes

1바이트 명령어 활용

Disassembly:

```
0:  b0 01                   mov    al,0x1
2:  89 c7                   mov    edi,eax
4:  b2 0e                   mov    dl,0xe
6:  e8 00 00 00 00          call   b <_main+0xb>
b:  5e                      pop    rsi
c:  40 b6 ff                mov    sil,0xff
f:  0f 05                   syscall
11: b0 3c                   mov    al,0x3c
13: 31 ff                   xor    edi,edi
15: 0f 05                   syscall
```

chunk1에 1바이트 공간이 생김

Best record **92** Bytes

1바이트 명령어 활용

Disassembly:

```
0:   b0 01                  mov    al,0x1
2:   89 c7                  mov    edi,eax
4:   b2 0e                  mov    dl,0xe
6:   e8 00 00 00 00         call   b <_main+0xb>
b:   5e                     pop    rsi
c:   40 b6 ff               mov    sil,0xff
f:   0f 05                  syscall
11:  b0 3c                  mov    al,0x3c
13:  31 ff                  xor    edi,edi
15:  0f 05                  syscall
```

chunk1에 0을 push하고
chunk4에 0을 pop하면
rdi를 0으로 만들 수 있음

Best record **92** Bytes

1바이트 명령어 활용

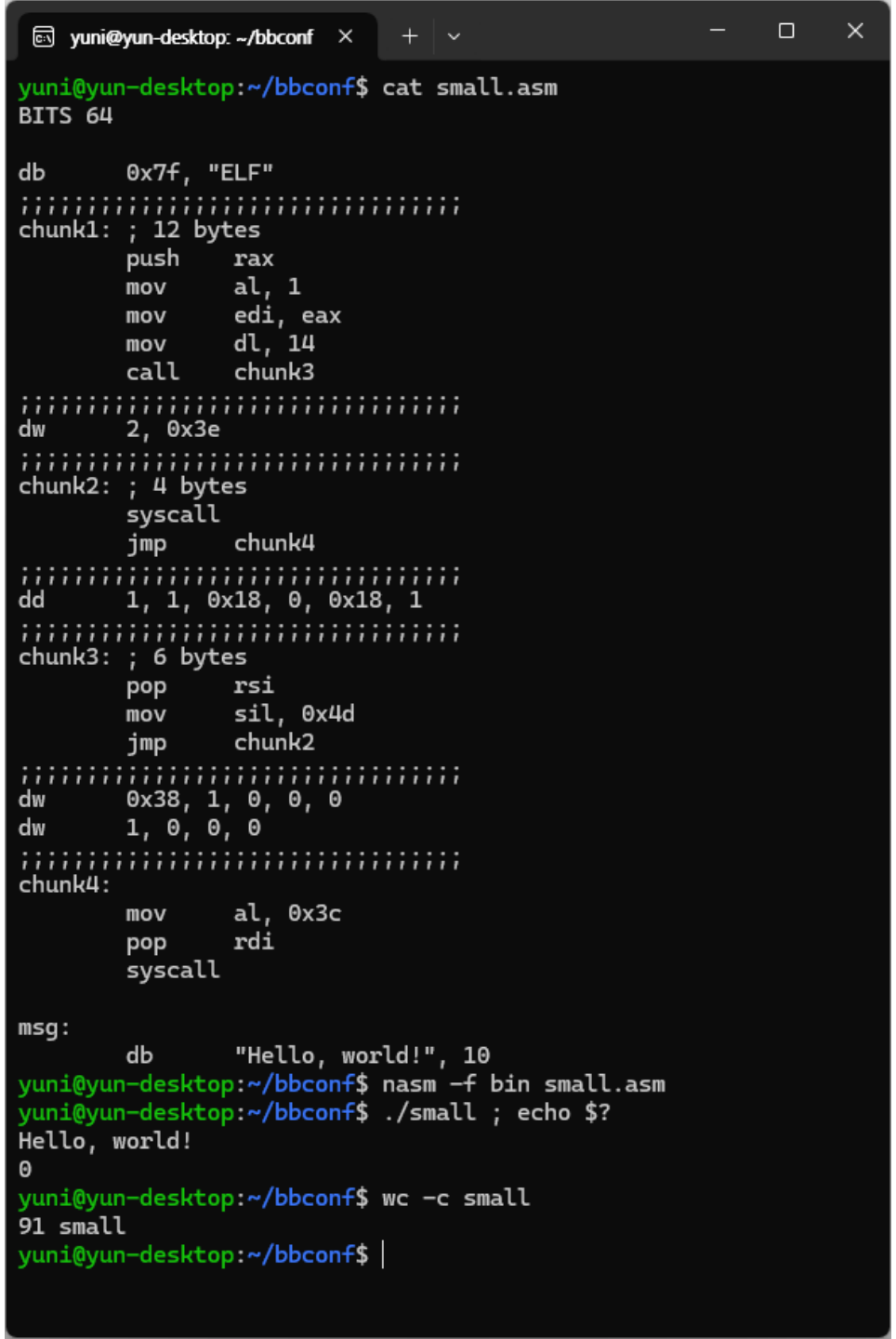
Disassembly:

```
0:  50                   push   rax
1:  b0 01                mov    al,0x1
3:  89 c7                mov    edi,eax
5:  b2 0e                mov    dl,0xe
7:  e8 00 00 00 00       call   c <_main+0xc>
c:  5e                   pop    rsi
d:  40 b6 ff             mov    sil,0xff
10: 0f 05                syscall
12: b0 3c                mov    al,0x3c
14: 5f                   pop    rdi
15: 0f 05                syscall
```

chunk1에 0을 push하고
chunk4에 0을 pop하면
rdi를 0으로 만들 수 있음

Best record **92** Bytes

1바이트 명령어 활용



Best record **92** -> **91** Bytes (▼1)

극한까지 활용한다

```
yuni@yun-desktop:~/bbconf$ xxd layout
00000000: 7f45 4c46 cccc cccc cccc cccc cccc cccc  .ELF............
00000010: 0200 3e00 cccc cccc 0100 0000 ddee 0000  ..>.............
00000020: 1800 0000 0000 0000 1800 0000 ddee 0000  ................
00000030: cccc cccc cccc 3800 0100 ffff ff00 0000  ......8.........
00000040: 0100 ffff ff00 0000 cccc cccc cccc cccc  ................
```

지금까지 0xdd, 0xee 필드는 활용하지 않았음

Best record **91** Bytes

극한까지 활용한다

```
yuni@yun-desktop:~/bbconf$ xxd layout
00000000: 7f45 4c46 cccc cccc cccc cccc cccc cccc  .ELF............
00000010: 0200 3e00 cccc cccc 0100 0000 ddee 0000  ..>.............
00000020: 1800 0000 0000 0000 1800 0000 ddee 0000  ................
00000030: cccc cccc cccc 3800 0100 ffff ff00 0000  ......8.........
00000040: 0100 ffff ff00 0000 cccc cccc cccc cccc  ................
```

지금까지 0xdd, 0xee 필드는 활용하지 않았음

jmp는 eb <offset> 꼴이므로, 0xdd 위치에 넣을 수 있음
동시에 앞의 4바이트 공간을 정숫값 1로 활용 가능

극한까지 활용한다

```
yuni@yun-desktop:~/bbconf$ xxd layout
00000000: 7f45 4c46 cccc cccc cccc cccc cccc cccc  .ELF............
00000010: 0200 3e00 cccc cccc 0100 0000 ddee 0000  ..>.............
00000020: 1800 0000 0000 0000 1800 0000 ddee 0000  ................
00000030: cccc cccc cccc 3800 0100 ffff ff00 0000  ......8.........
00000040: 0100 ffff ff00 0000 cccc cccc cccc cccc  ................
```

지금까지 0xdd, 0xee 필드는 활용하지 않았음

jmp는 eb <offset> 꼴이므로, 0xdd 위치에 넣을 수 있음
동시에 앞의 4바이트 공간을 정숫값 1로 활용 가능

또한 call뿐만 아니라 syscall의 경우에도
다음 명령어 주소가 rcx 레지스터에 저장됨을 활용

극한까지 활용한다

Disassembly:

```
0:  50                      push   rax
1:  0f 05                   syscall
3:  b0 01                   mov    al,0x1
5:  b1 48                   mov    cl,0x48
7:  51                      push   rcx
8:  b2 0e                   mov    dl,0xe
a:  e9 00 00 00 00          jmp    f <_main+0xf>
f:  5e                      pop    rsi
10: b3 3c                   mov    bl,0x3c
12: bf 01 00 00 00          mov    edi,0x1
17: e9 00 00 00 00          jmp    1c <_main+0x1c>
1c: 0f 05                   syscall
1e: 93                      xchg   ebx,eax
1f: 5f                      pop    rdi
20: 0f 05                   syscall
```

최종 버전

Best record **91** Bytes

극한까지 활용한다

Disassembly:

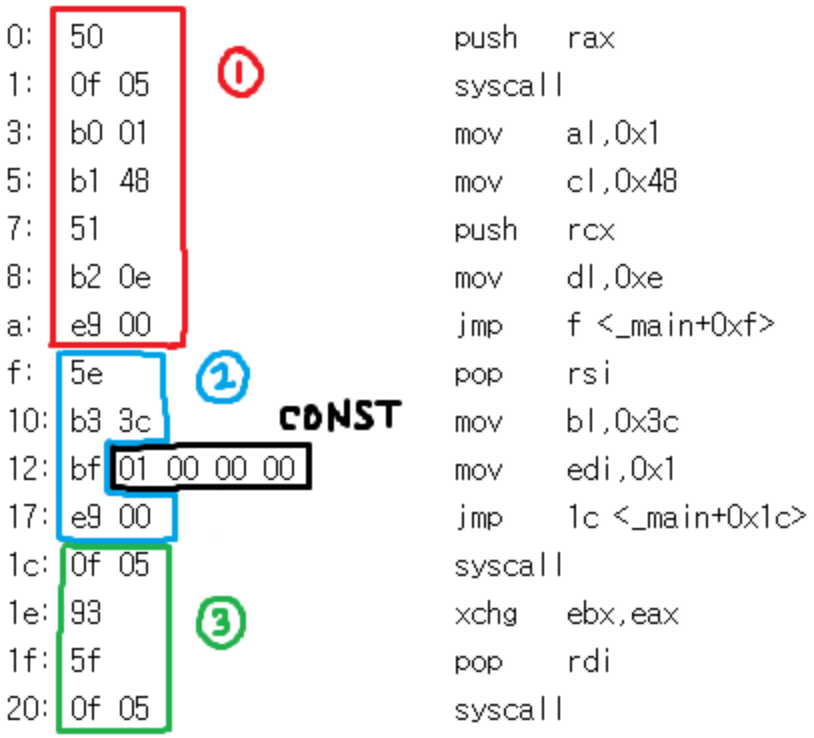
```
0:  50                      push   rax
1:  0f 05                   syscall
3:  b0 01                   mov    al,0x1
5:  b1 48                   mov    cl,0x48
7:  51                      push   rcx
8:  b2 0e                   mov    dl,0xe
a:  e9 00                   jmp    f <_main+0xf>
f:  5e                      pop    rsi
10: b3 3c                   mov    bl,0x3c
12: bf 01 00 00 00          mov    edi,0x1
17: e9 00                   jmp    1c <_main+0x1c>
1c: 0f 05                   syscall
1e: 93                      xchg   ebx,eax
1f: 5f                      pop    rdi
20: 0f 05                   syscall
```

jmp 명령어의 offset이 이상하게 나와서 수정

Best record **91** Bytes

극한까지 활용한다

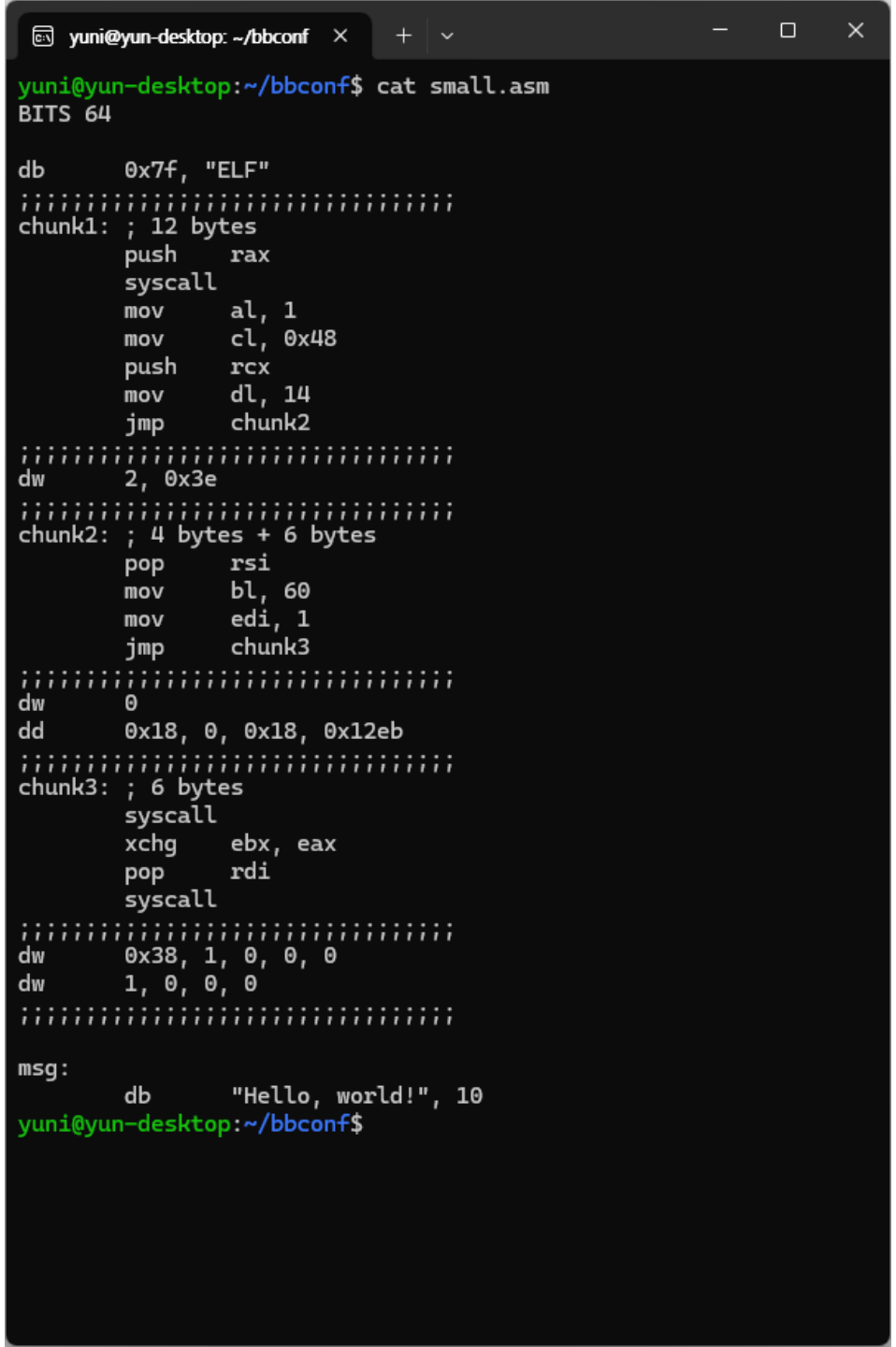
Disassembly:



더 이상 chunk4 자리가 필요 없어짐

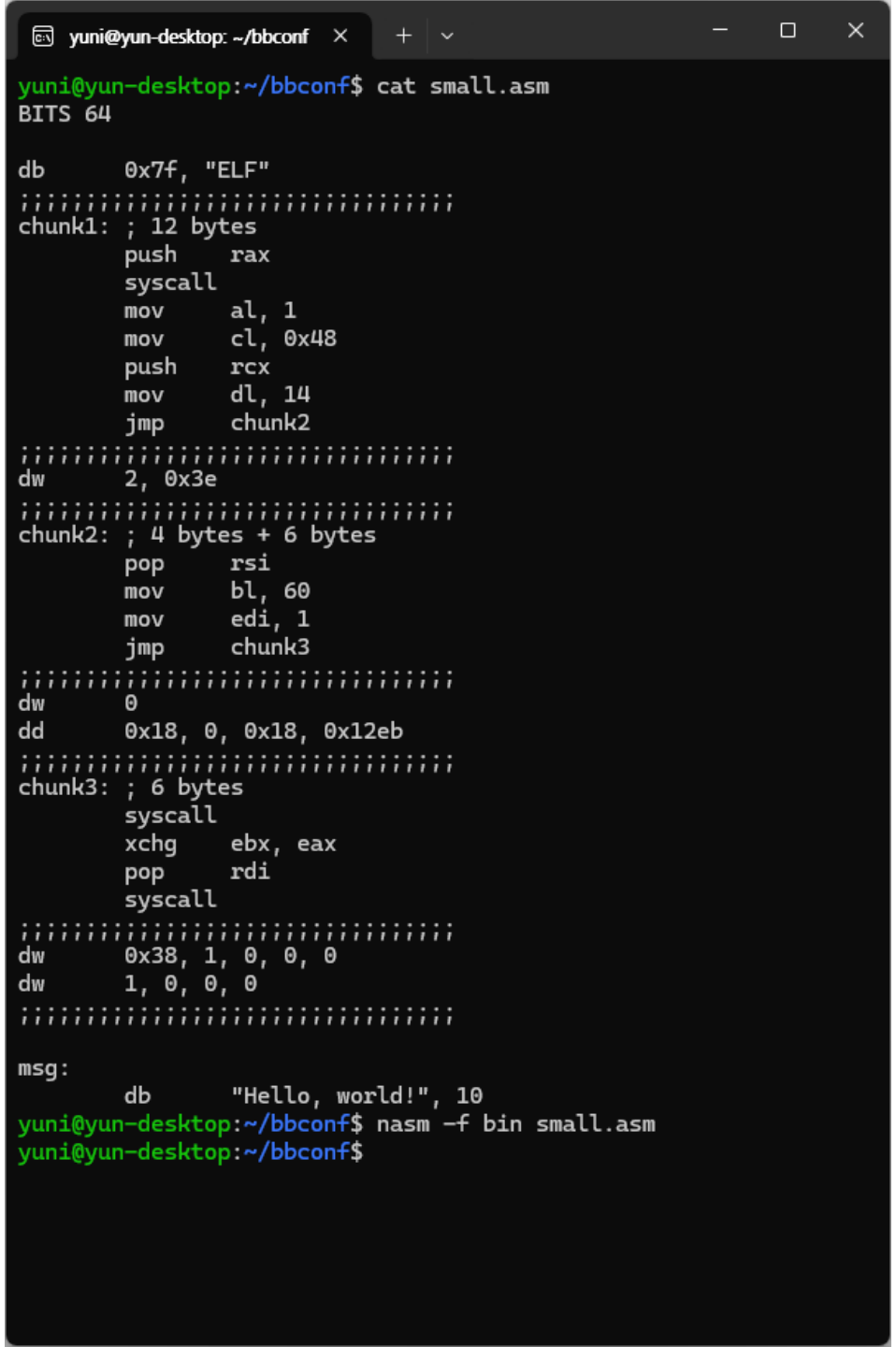
Best record **91** Bytes

극한까지 활용한다



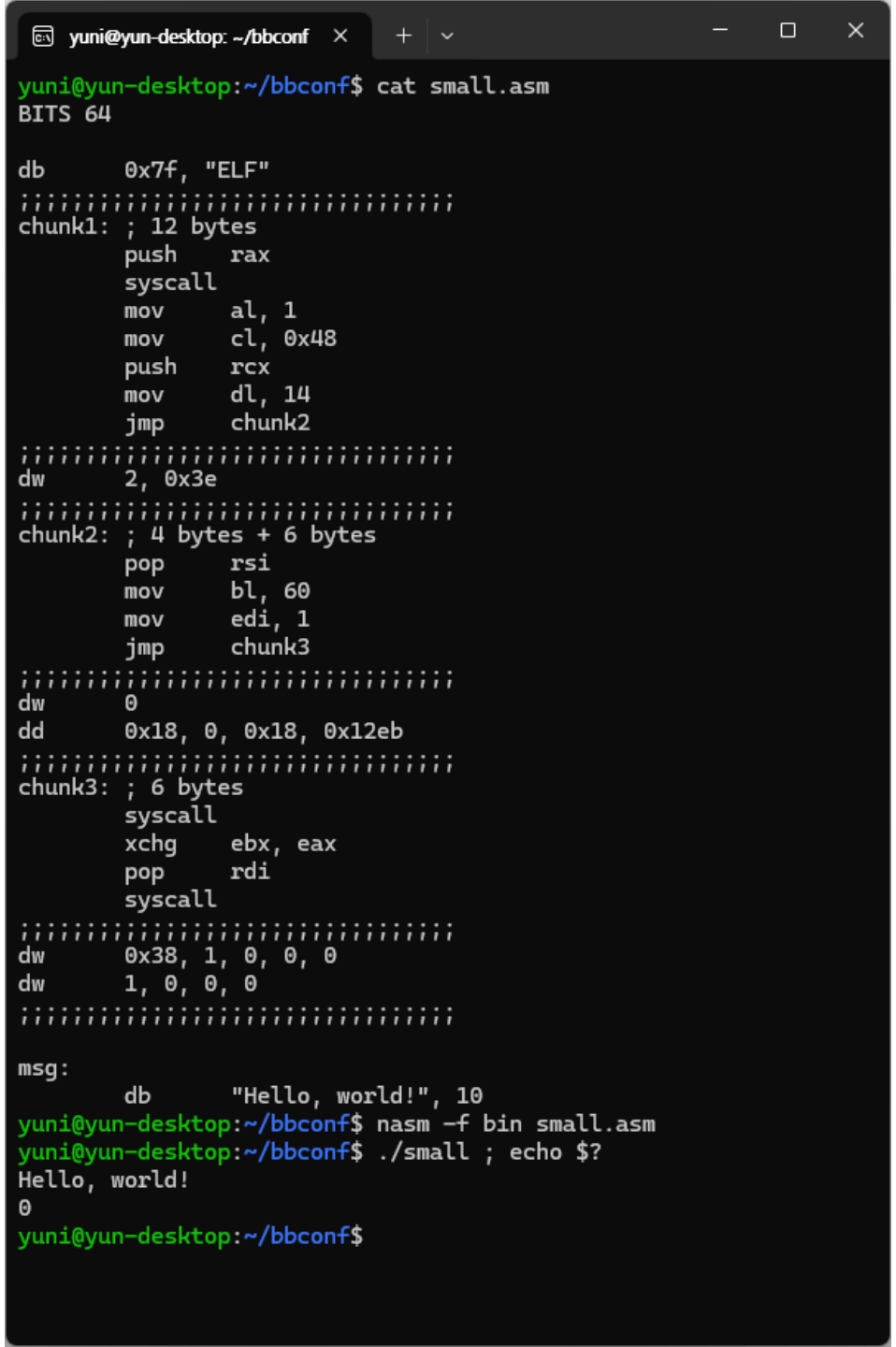
Best record **91** Bytes

극한까지 활용한다



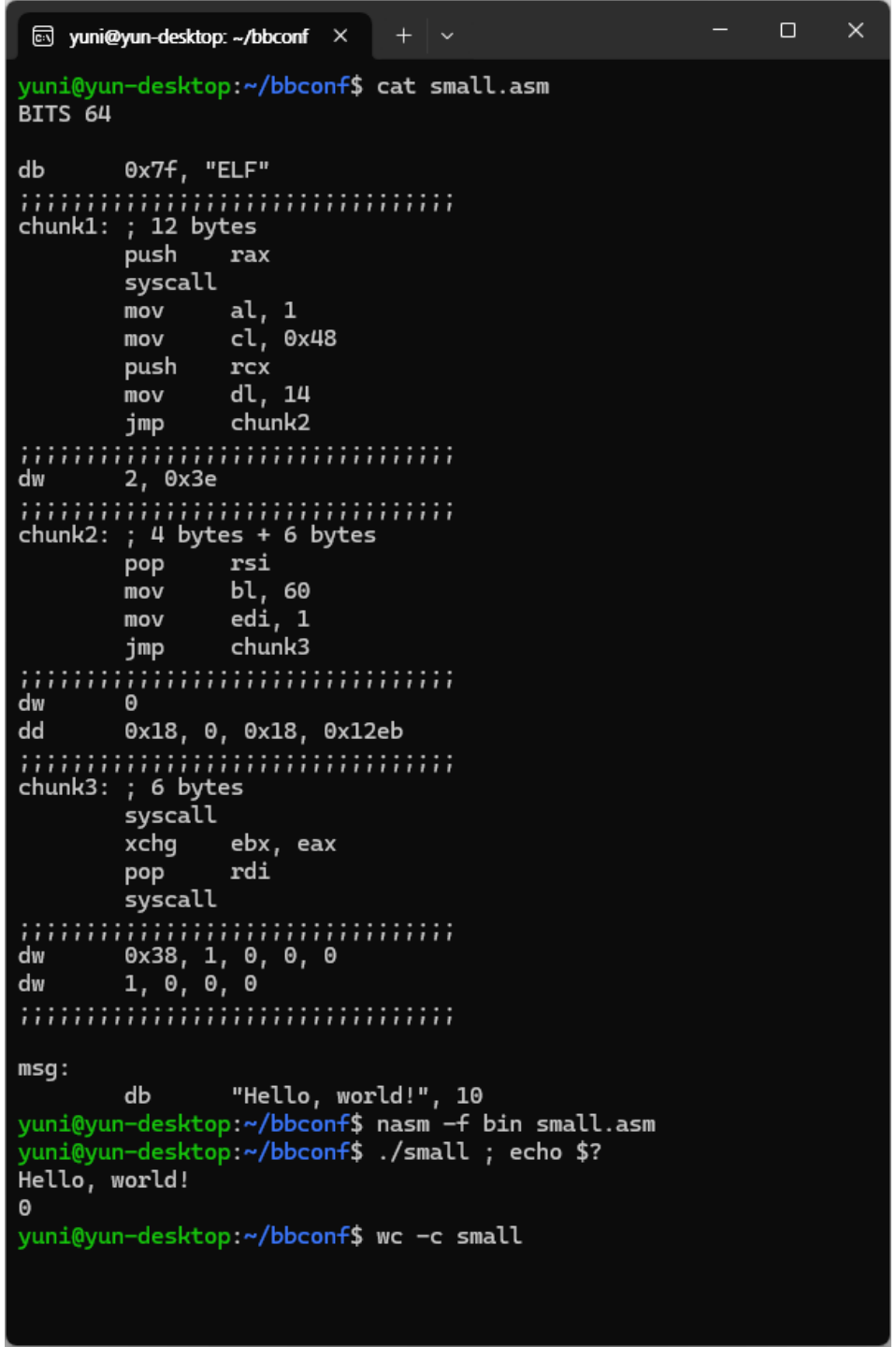
Best record **91** Bytes

극한까지 활용한다



Best record **91** Bytes

극한까지 활용한다



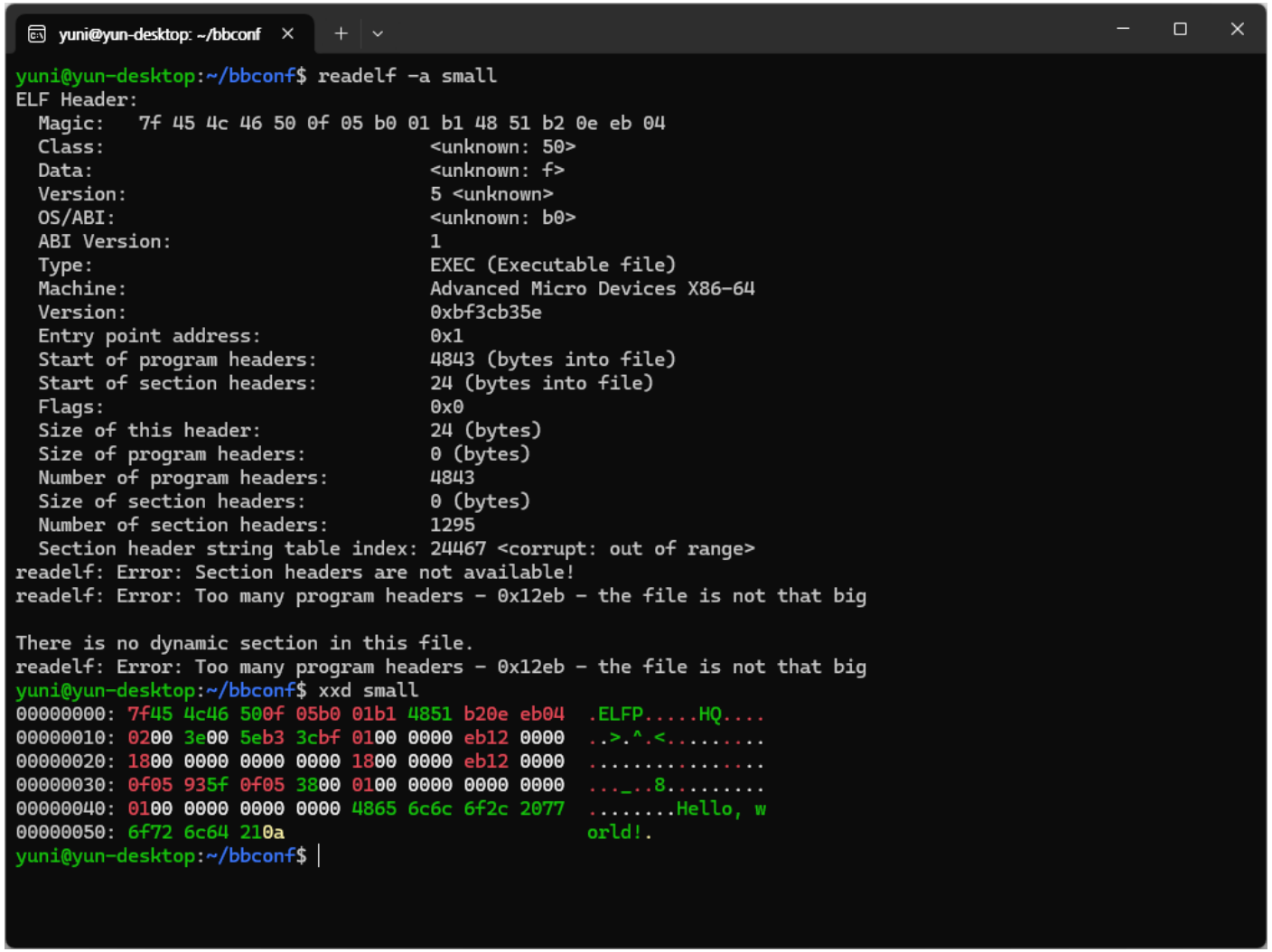
Best record **91** Bytes

극한까지 활용한다

```
yuni@yun-desktop:~/bbconf$ cat small.asm
BITS 64

db      0x7f, "ELF"
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
chunk1: ; 12 bytes
        push    rax
        syscall
        mov     al, 1
        mov     cl, 0x48
        push    rcx
        mov     dl, 14
        jmp     chunk2
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
dw      2, 0x3e
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
chunk2: ; 4 bytes + 6 bytes
        pop     rsi
        mov     bl, 60
        mov     edi, 1
        jmp     chunk3
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
dw      0
dd      0x18, 0, 0x18, 0x12eb
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
chunk3: ; 6 bytes
        syscall
        xchg    ebx, eax
        pop     rdi
        syscall
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
dw      0x38, 1, 0, 0, 0
dw      1, 0, 0, 0
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

msg:
        db      "Hello, world!", 10
yuni@yun-desktop:~/bbconf$ nasm -f bin small.asm
yuni@yun-desktop:~/bbconf$ ./small ; echo $?
Hello, world!
0
yuni@yun-desktop:~/bbconf$ wc -c small
86 small
yuni@yun-desktop:~/bbconf$
```

Best record **91** -> **86** Bytes (▼5)

최종 결과물



Best record **86** Bytes

결론

헤더를 최대한 겹쳤고,
겹친 헤더의 활용 가능한 필드에 머신 코드를 주입했고,
헤더의 끝부터 바로 출력할 데이터를 넣었으므로,
(14바이트는 연속되어야 하기 때문에 헤더 속에 위치할 수 없음)

이론상 헬로월드 출력 프로그램의 최소 크기는 **86바이트**

Best record **86** Bytes

결론

여러분이 만든 프로그램은,
얼마나 많은 바이트를 낭비하고 있나요?

Best record **86** Bytes

Thank you for watching