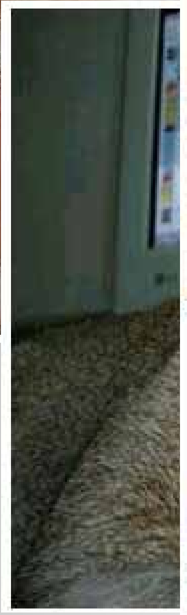
Calling Convention & Disassembling

THE ENDLESS PLAY

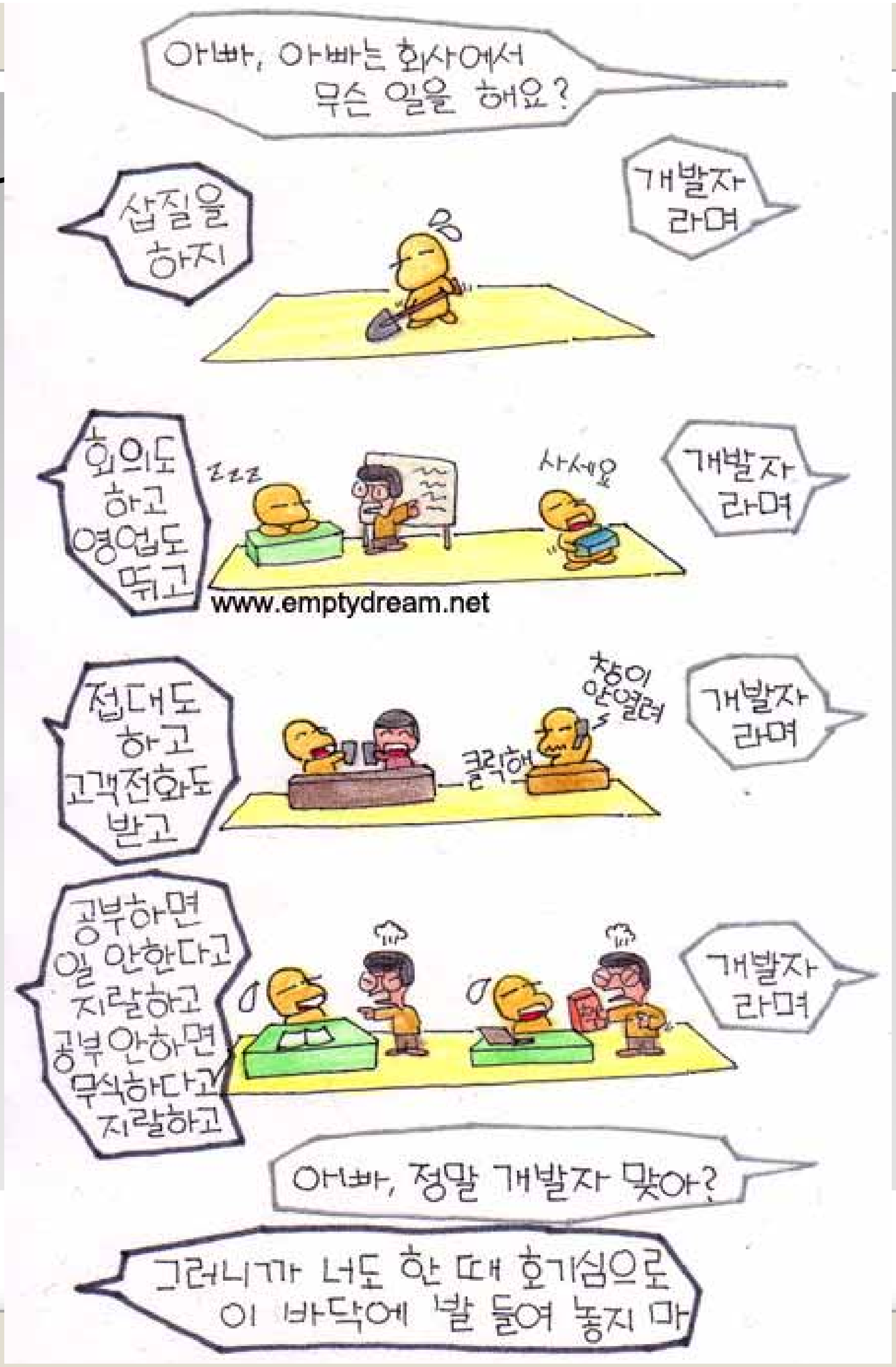
2007 . 7 . 21
gurugio @ asmlove



기오 소

Code Engn
http://www.CodeEngn.com

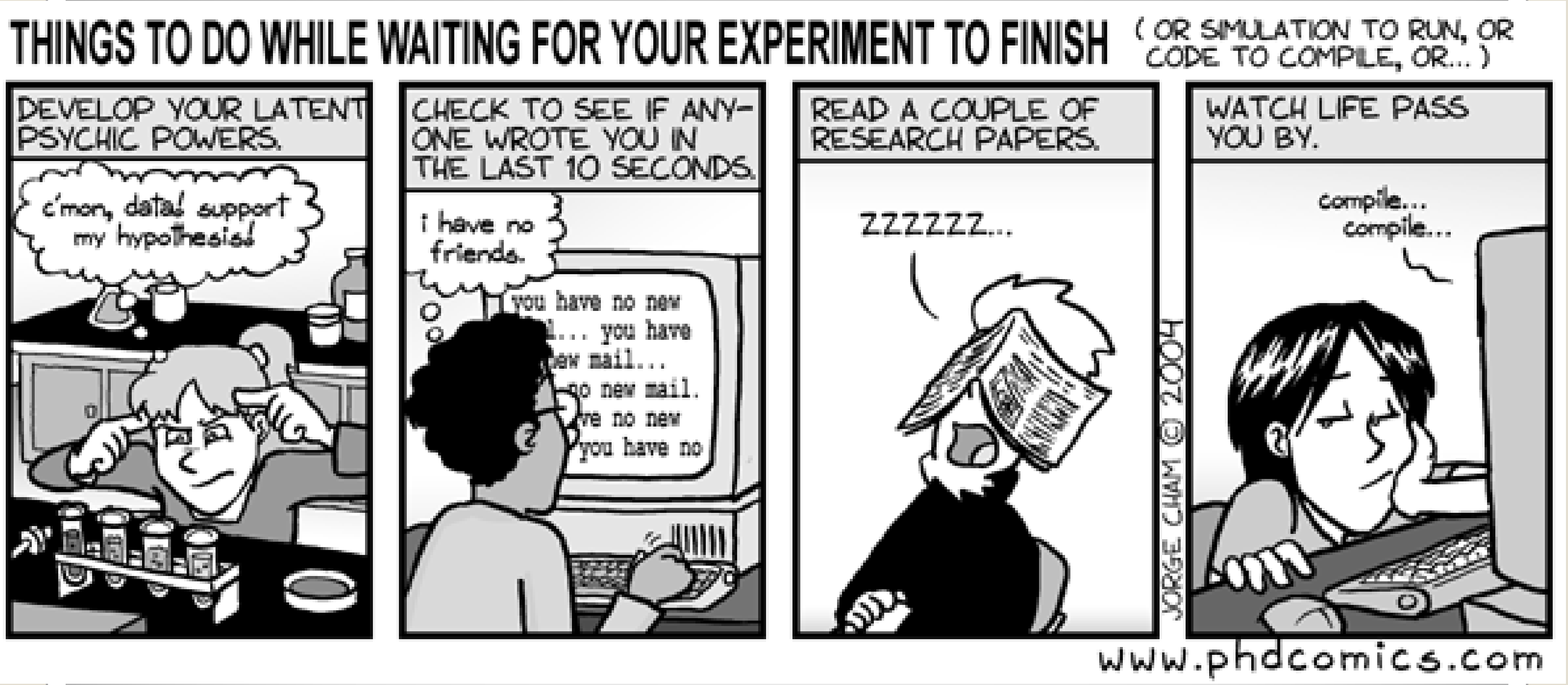
- 이러고 삽니


아빠, 아빠는 회사에서 무슨 일을 해요?
삽질을 하지
개발자라며
회의도 하고 영업도 뛰고
zzz
사세요
개발자라며
www.emptydream.net
접대도 하고 고객전화도 받고
창이 안열려
클릭해
개발자라며
공부하면 일 안한다고 지랄하고 공부 안하면 무식하다고 지랄하고
개발자라며
아빠, 정말 개발자 맞아?
그러니까 너도 한 때 호기심으로 이 바닥에 발 들여 놓지 마

기오 소개

Code Engn
http://www.CodeEngn.com

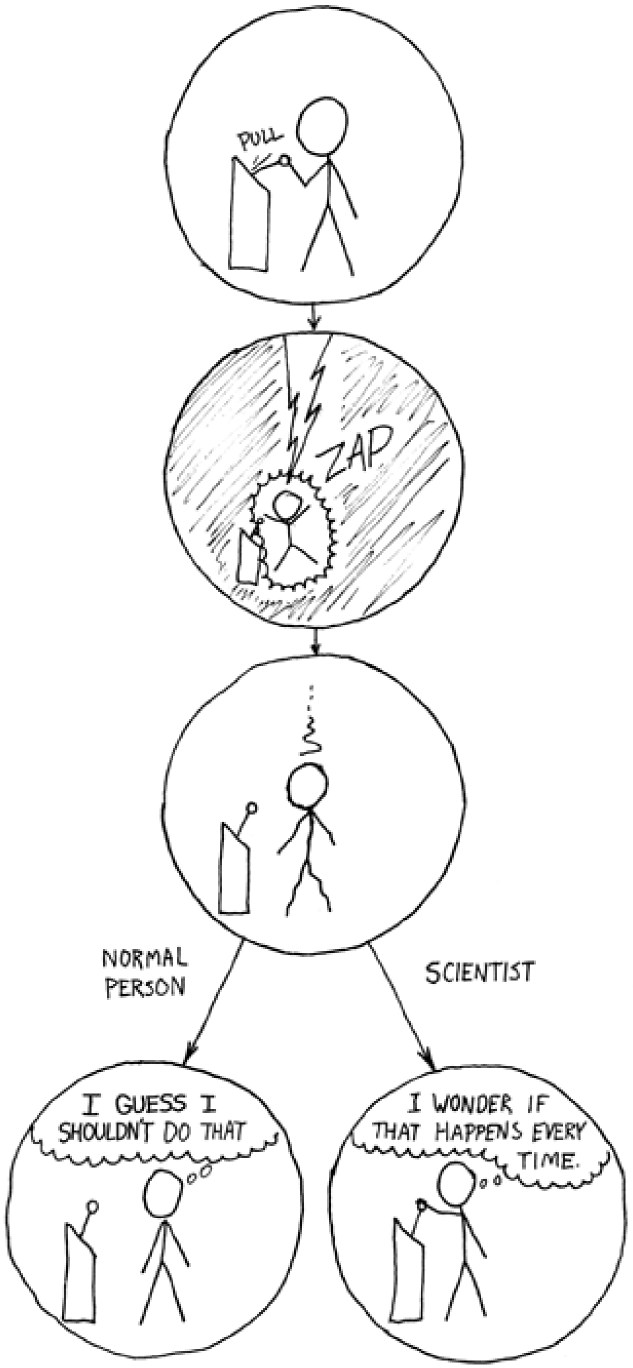
취미 생활


THINGS TO DO WHILE WAITING FOR YOUR EXPERIMENT TO FINISH
(OR SIMULATION TO RUN, OR CODE TO COMPILE, OR...)
DEVELOP YOUR LATENT PSYCHIC POWERS.
c'mon, data! support my hypothesis!
CHECK TO SEE IF ANYONE WROTE YOU IN THE LAST 10 SECONDS.
i have no friends.
you have no new mail... you have no new mail... you have no new mail. you have no new mail. you have no
READ A COUPLE OF RESEARCH PAPERS.
ZZZZZZ...
WATCH LIFE PASS YOU BY.
compile... compile...
JORGE CHAM © 2004
www.phdcomics.com

기오 소개

Code Engn
http://www.CodeEngn.com

- 성격


PULL
ZAP
NORMAL PERSON
SCIENTIST
I GUESS I SHOULDN'T DO THAT
I WONDER IF THAT HAPPENS EVERY TIME.

기오 소개

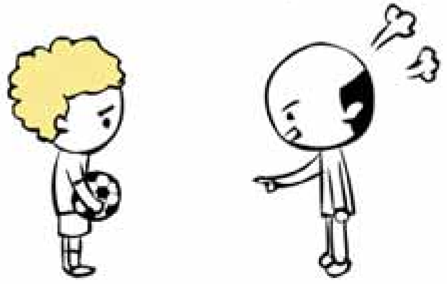
Code Engn
http://www.CodeEngn.com

Yamako© 18

공감만화-4.내이름은 세브첸코

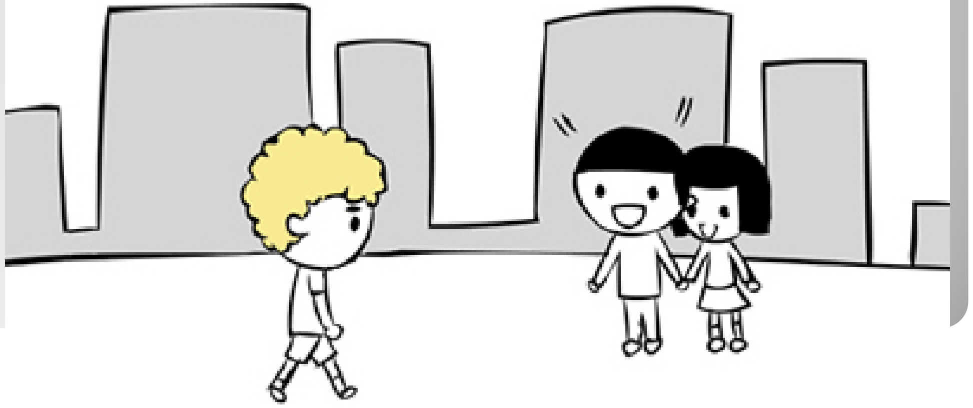
내 이름은 세브첸코 내 얘기한번 들어 볼래?
난 가끔 아직 학생이던 어린시절을 떠올려
그때로 돌아간다면 어떨까?

내 꿈은 의대에 진학해서 의사가 되는거였어
하지만 난 공부보다 축구를 더 많이 했지



결국 성적이 떨어져 의대진학에 실패했고
난 정식으로 축구선수가 되었어

명문팀에 입단후 곧 국가대표팀에 뽑혔고,
어딜가나 사람들은 날 '득점기계'라 불렀어



• 공부를 열심히~~

내 별명은 '득점기계'
맞아 난 사람대접도 제대로 못받고있어...
선생님 말씀대로 공부만 열심히 했더라면...
'공부' 그것은 때가 있는거다. Fin.

기오

Code⚡Engn
http://www.CodeEngn.com

C

http:/

잠 깐 잡 담

- 삶이 지루해요
- 시간이 남아서 벽지 무늬 개수를 세어봤어요
- 여친이 없어서 주말이 두려워요
- 밤이 무섭고 잠이 안와요
- …
- 어셈블리 코딩을 해보세요.
- 여러분의 고민이 해결됩니다.

뭔가 기대되죠?

- 어셈블리?
  - 기계어인가?
  - 옛날에 쓰던 고어(古語 or 苦語)?
  - 골치아픈 숙제
  - 우리나라에서도 쓰는 사람이 있나?
  - 해커 나오는 영화에서 본거 같당
  - …

이런 사람 있어요~

- Really?
  - 컴퓨터를 깊게 이해하게 됩니다.
  - 프로그래밍 언어를 깊게 이해하게 됩니다.
  - 특정 플랫폼이 제공하는 강력한 기능을 사용할 수 있습니다.
  - 가끔은 더 빠르게 실행되는 프로그램도 만들 수 있습니다.
  - 저는 OS 커널 만들기 공부를 시작할 수 있었습니다.

뭔가 기대되죠?

- 이렇게 됩니다!! 진짜?

마감 준수 세포
UI 개선의지
오늘 너무 심심한데 잠깐 코딩이나 해볼까?
열정
꼼꼼한 디버깅
정 불편하면 니가 만들던가
C&P
열정 퇴화 중추
나도 돈 벌고싶다
계획적인 개발
검색 검색 검색 검색 검색
이름 남기기 뉴런
하루 12시간 작업? KIN
사용자의 불편을 최소화 해주겠다는 의지
강력한 기업에 대한 반발심
'안되면 되게하라' 중추
오픈소스 개발자의 뇌 구조

Calling Convention

- Calling Convention
  - 함수에 값을 전달하고 함수가 끝난 후 복귀하는 과정을 정의한 하나의 규칙
  - 언어마다 다름
  - 어셈블리 코드를 만들 때는 같이 사용할 고급 언어의 Calling Convention을 잘 알아야 함
- Fast Calling Convention
- Pascal Calling Convention
- C Calling Convention
- 왜 C 함수는 인자가 몇 개인지 표시하지 않을까요?

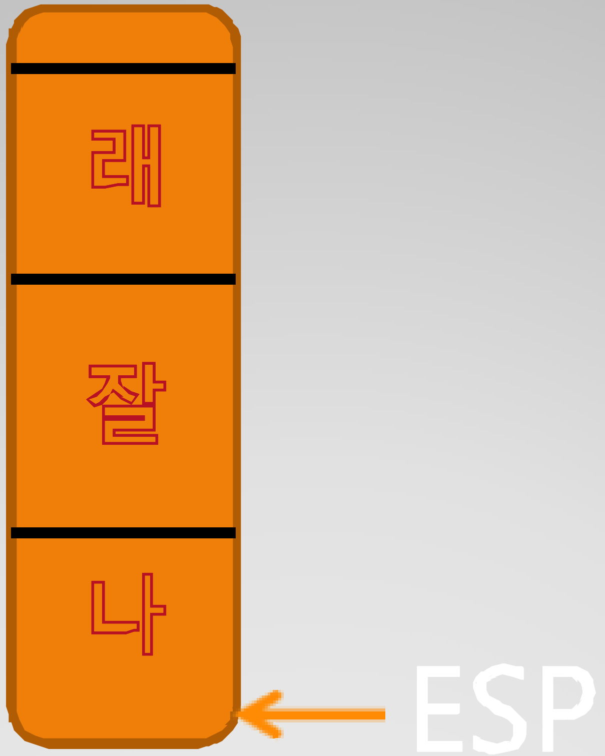
슬슬 졸림!

- 미리 알아두어야 할 것

- PUSH
  - 스택 포인터 옮기기
  - 데이터 저장
- POP
  - 데이터 읽기
  - 스택 포인터 옮기기
- **스택은 항상 4바이트 단위로 옮겨짐**
- **32비트 머신이니까!**

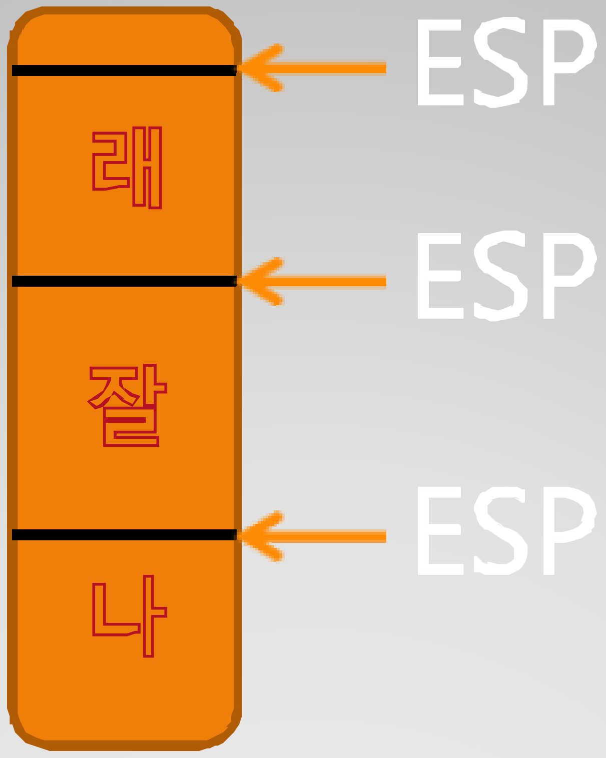
나 잘래~

PUSH



이것만은 기억해주세요~ (굽신굽신~)

POP

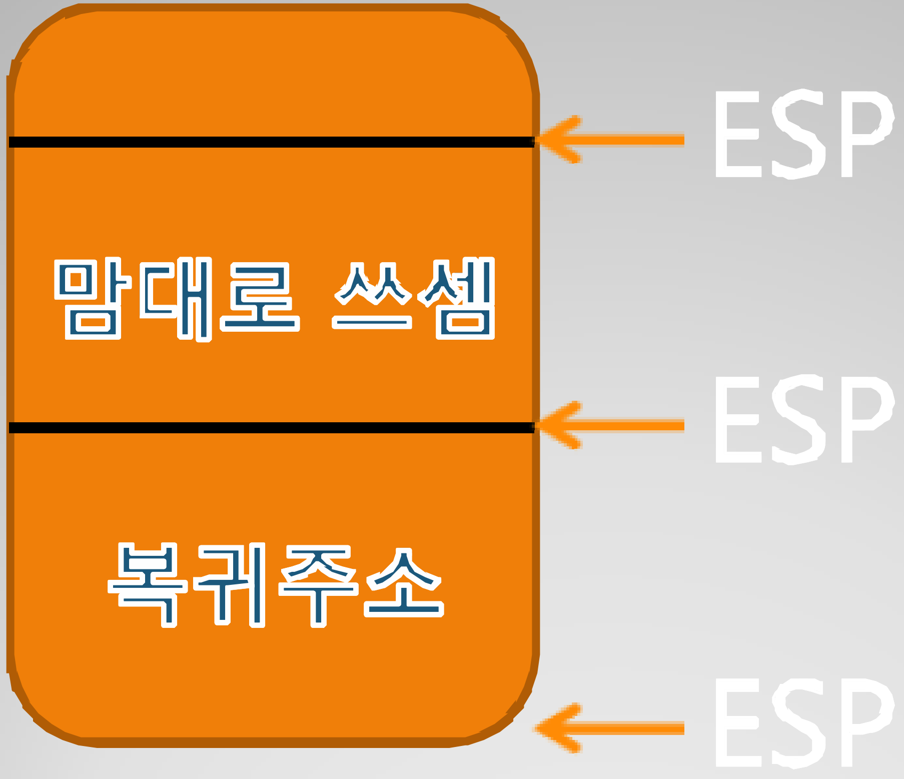


이것만은 기억해주세요~ (굽신굽신~)

- Call
  - 다른 함수로 갔다올께염
  - 돌아올 곳은 남겨놔야지~
- RET
  - 나 다시 돌아갈래~~
  - 근데 어디로 가야하나?
- 스택이 중요한 이유 아시겠지요?

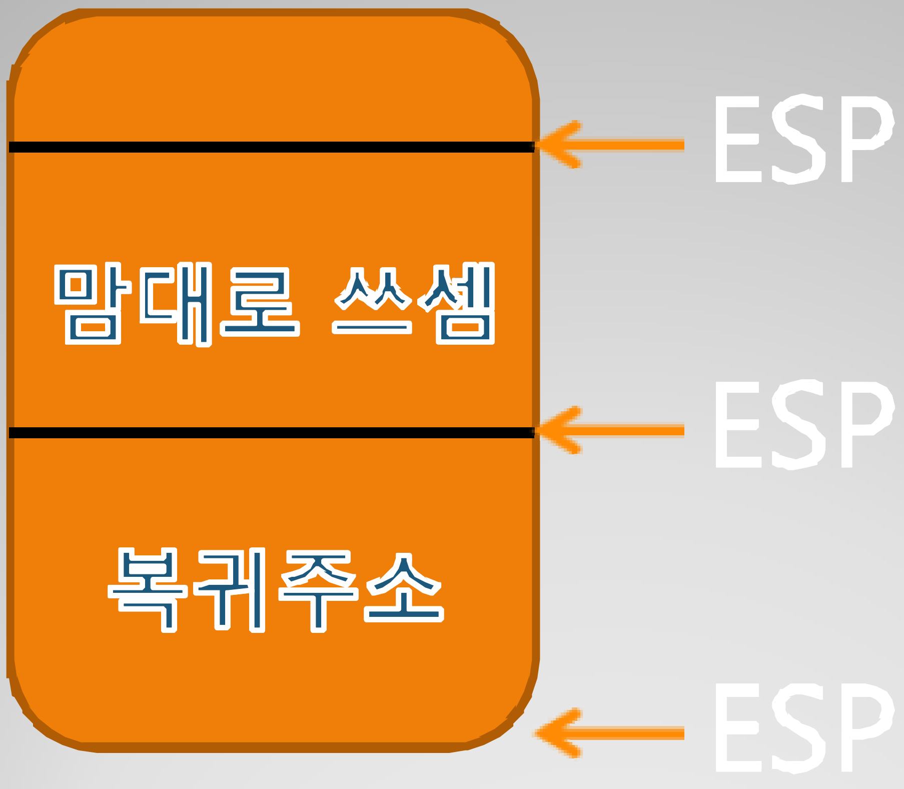
이것만은 기억해주세요~ (굽신굽신~)

- 갈때 가더라도 돌아올 곳을 남겨놓자



이것만은 기억해주세요~ (굽신굽신~)

- 떠날 때는 빈손으로



이것만은 기억해주세요~ (굽신굽신~)

- 만약 스택이 꼬이면?
- 엉뚱한 곳으로 점프~
- 엉뚱한 코드가 실행되고
- 나쁜 코드가 실행될지도 모름
- 사실은 알고싶은게 이거? @_@
- 저도 잘 몰라요
- 실수로 Bomb을 만들어 본 적은 있음 ^^;
- 이거 만들어달라는 사람 꼭 있음 -_-+

이것만은 기억해주세요~ (굽신굽신~)

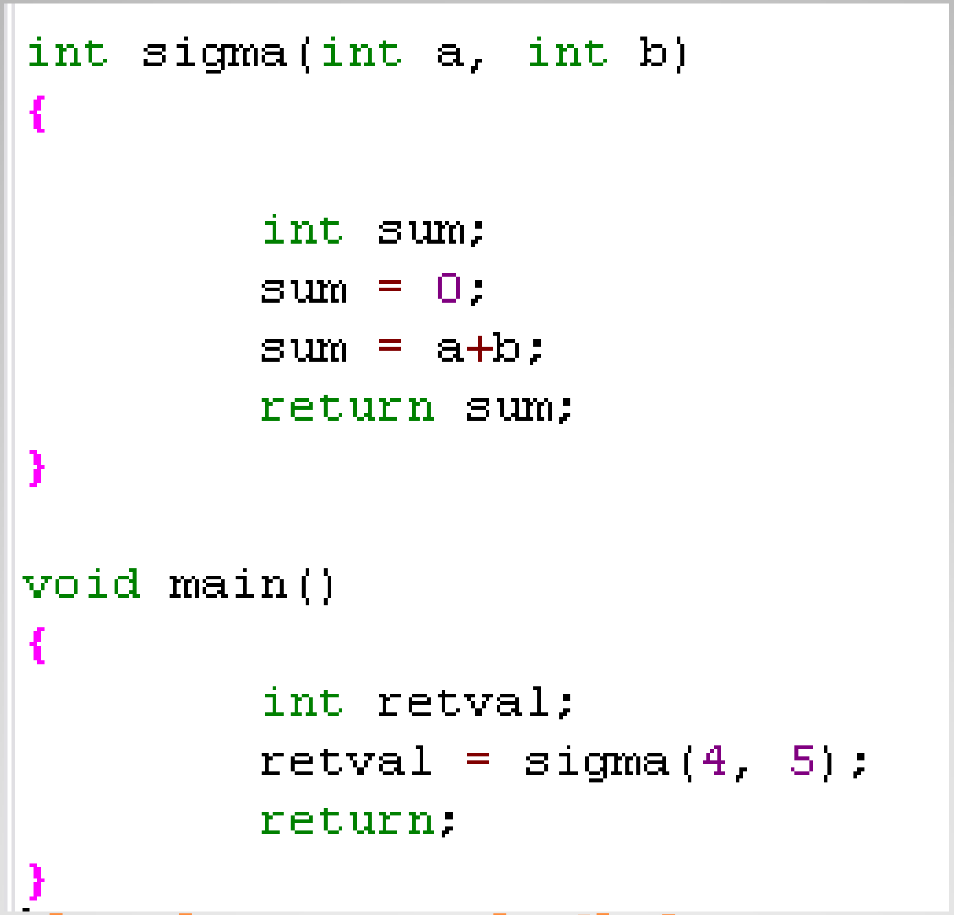
- C Calling Convention 의 특징
- 함수를 호출하기 전에
- 함수 인자를 스택에 저장한다.
- 함수가 끝날 때는
- 함수 호출 직전 스택으로 복귀
- 함수가 끝난 후 함수 인자를 지운다

예제를 같이 읽어 보아요

이것만은 기억해주세요~ (굽신굽신~)

Calling Convention Example

그냥 짧은 C 함수.. 그러나..



```
int sigma(int a, int b)
{

        int sum;
        sum = 0;
        sum = a+b;
        return sum;
}

void main()
{
        int retval;
        retval = sigma(4, 5);
        return;
}
```

여기부터 쬐끔 복잡해효~

뭐야 이거..몰라..무서워..

```
_sigma PROC NEAR

>>> int sum;
        push ebp
        mov ebp, esp
        sub esp, 4
>>> sum = 0;

        mov DWORD PTR [ebp-4], 0

>>> sum = a + b;
        mov eax, dword ptr [ebp+8]
        add eax, dword ptr [ebp+12]

        mov dword ptr [ebp-4], eax

>>> return sum
```

짧게 핵심만 살펴봅시다
전체 소스는 어셈러브로 GO!GO!

```
_main proc near
        push ebp
        mov ebp, esp
        push ecx

        push 5
        push OFFSET FLAT:_array
        call _sigma
        add esp, 8
        and eax, 65536
        mov dword ptr _retval$[ebp], eax

        mov esp, ebp
        pop ebp
        ret 0
_main endp
```

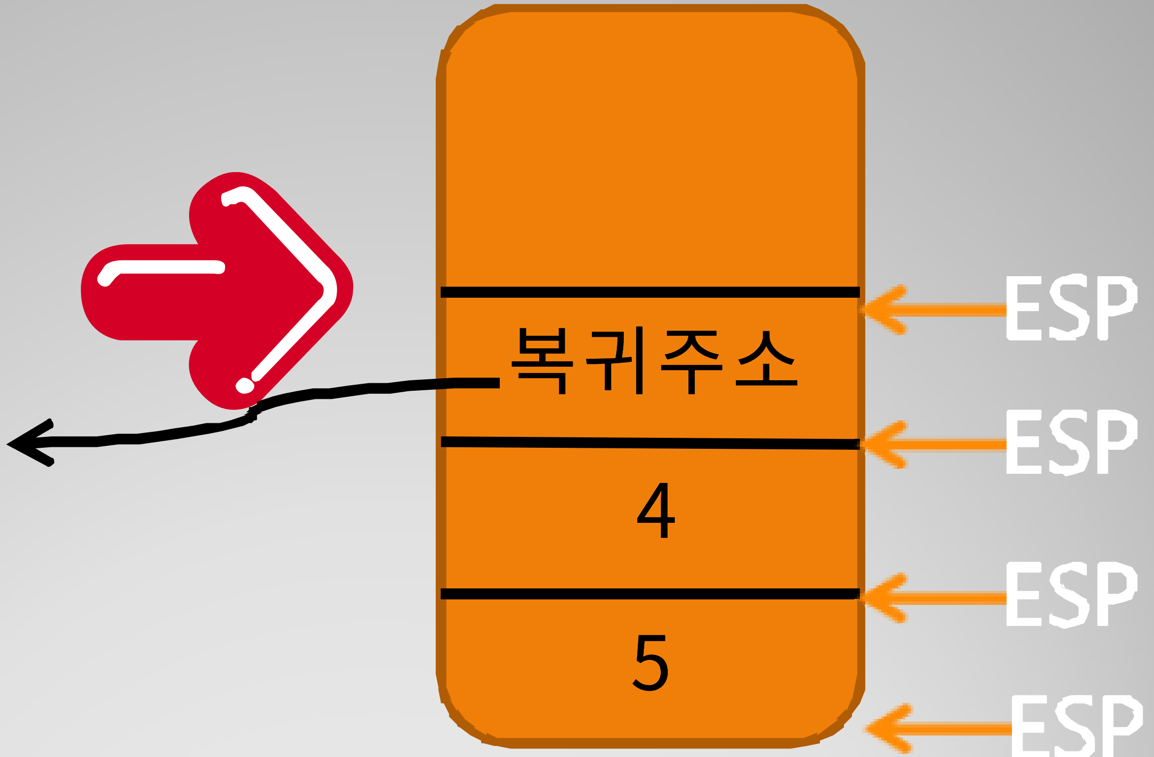
```
retval = sigma(4, 5);
```

```
push 5
push 4
call sigma
add esp, 8
```

뒤부터!! 왜?

여기

```
push 5
push 4
call sigma
add esp, 8
```



여기부턴 찌끔 복잡해효~

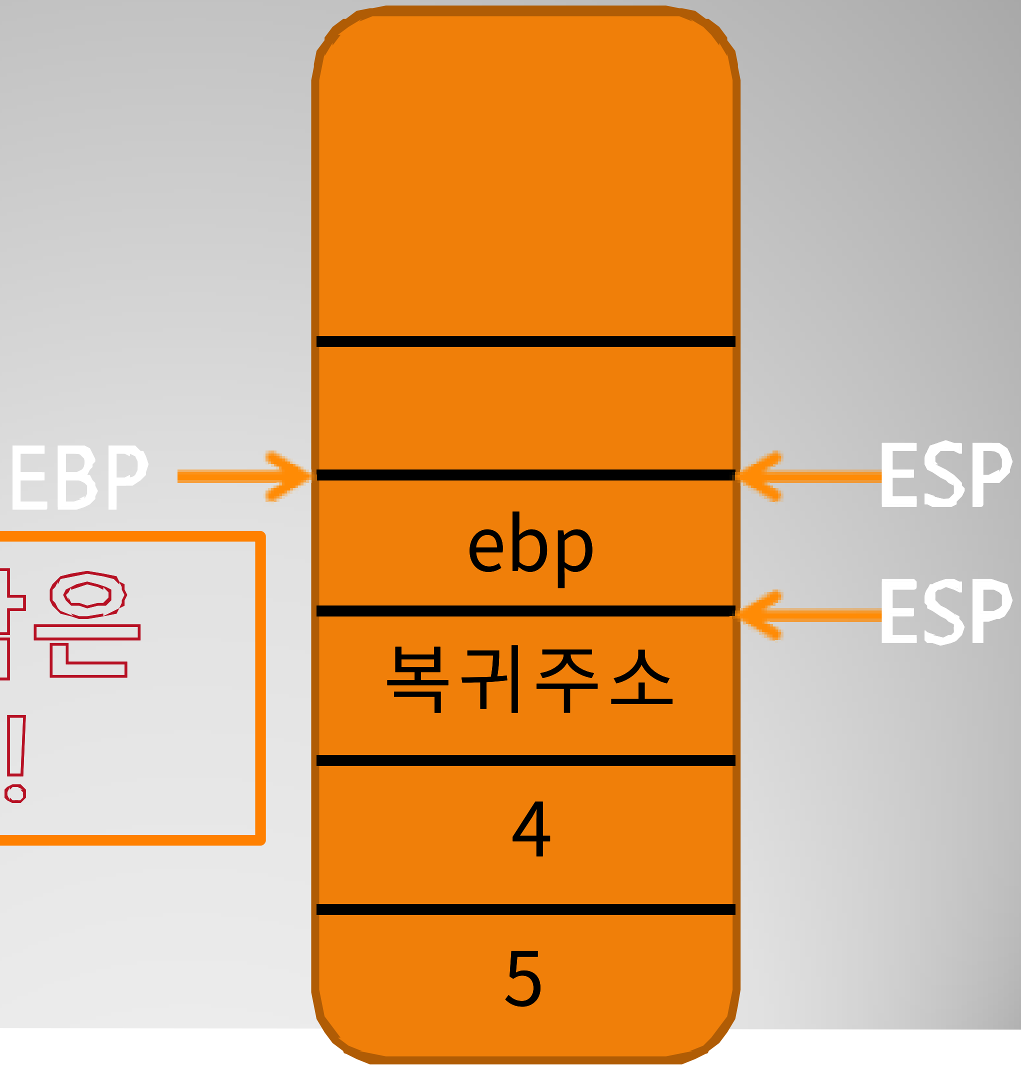
```
_sigma PROC NEAR
        push ebp
        mov ebp, esp

>>> int sum;
        sub esp, 4

>>> sum = 0;

        mov DWORD PTR [ebp-4], 0

>>> sum = a + b;
        mov eax, dword ptr [ebp+8]
        add eax, dword ptr [ebp+12]

        mov dword ptr [ebp-4], eax

>>> return sum;
        mov eax, dword ptr [ebp-4]

        mov esp, ebp
        pop ebp

        ret
_sigma ENDP
```

작게 나눠 정복하기!

Divide & Conquer!

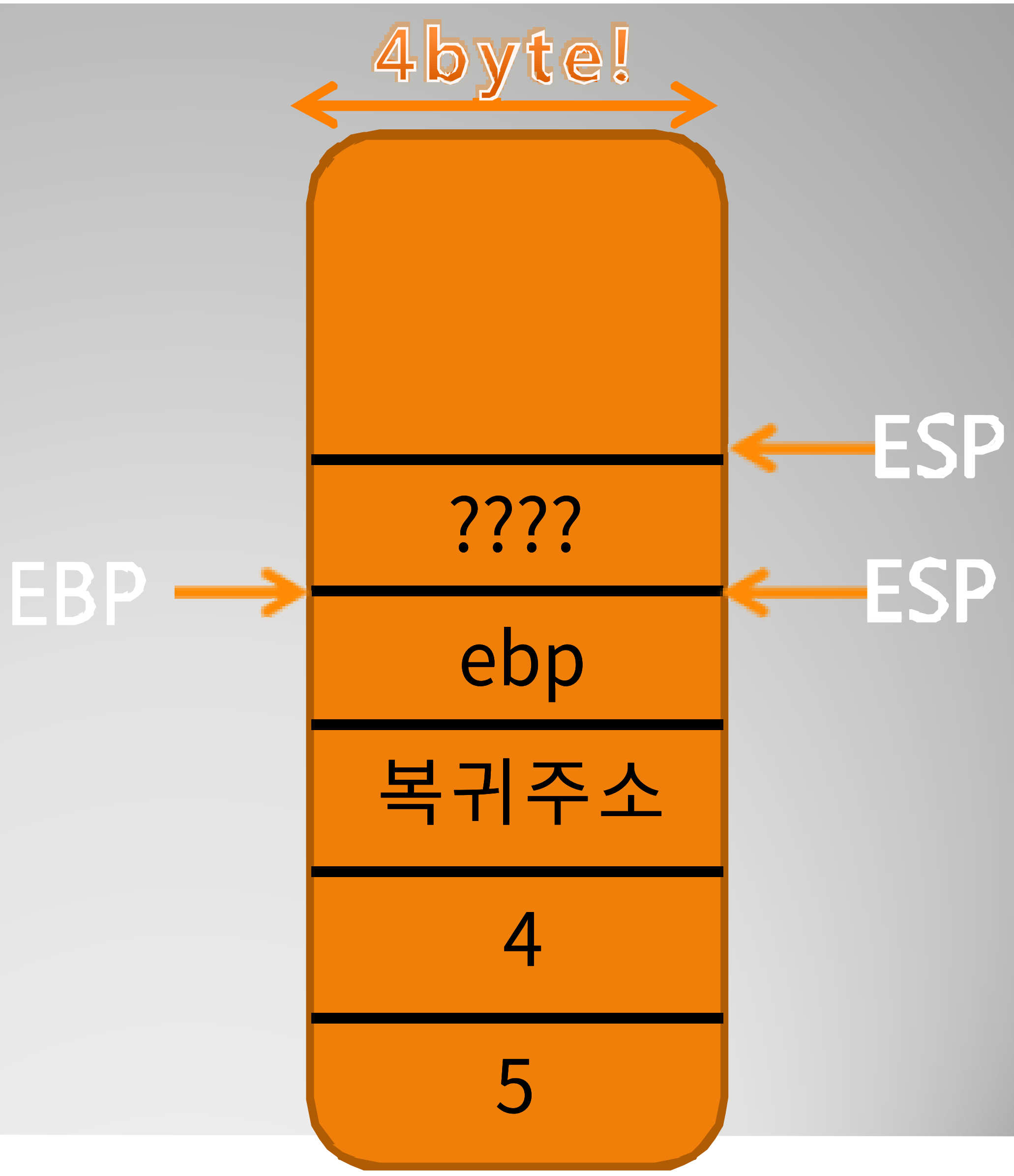
여기부턴 쬐끔 복잡해효~

```
_sigma PROC  NEAR
        push ebp
        mov ebp, esp
```

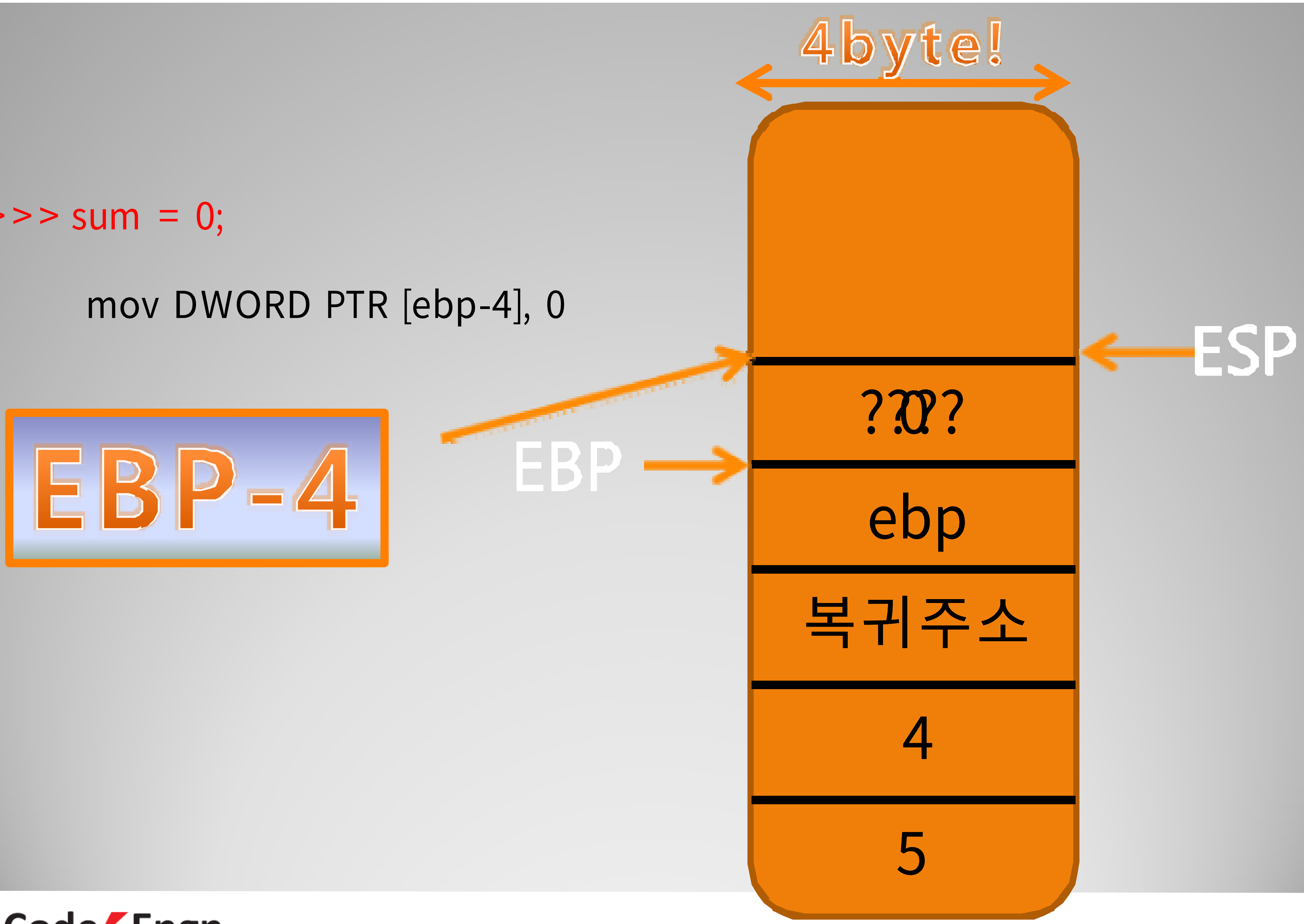


함수의 시작은
EBP 저장!

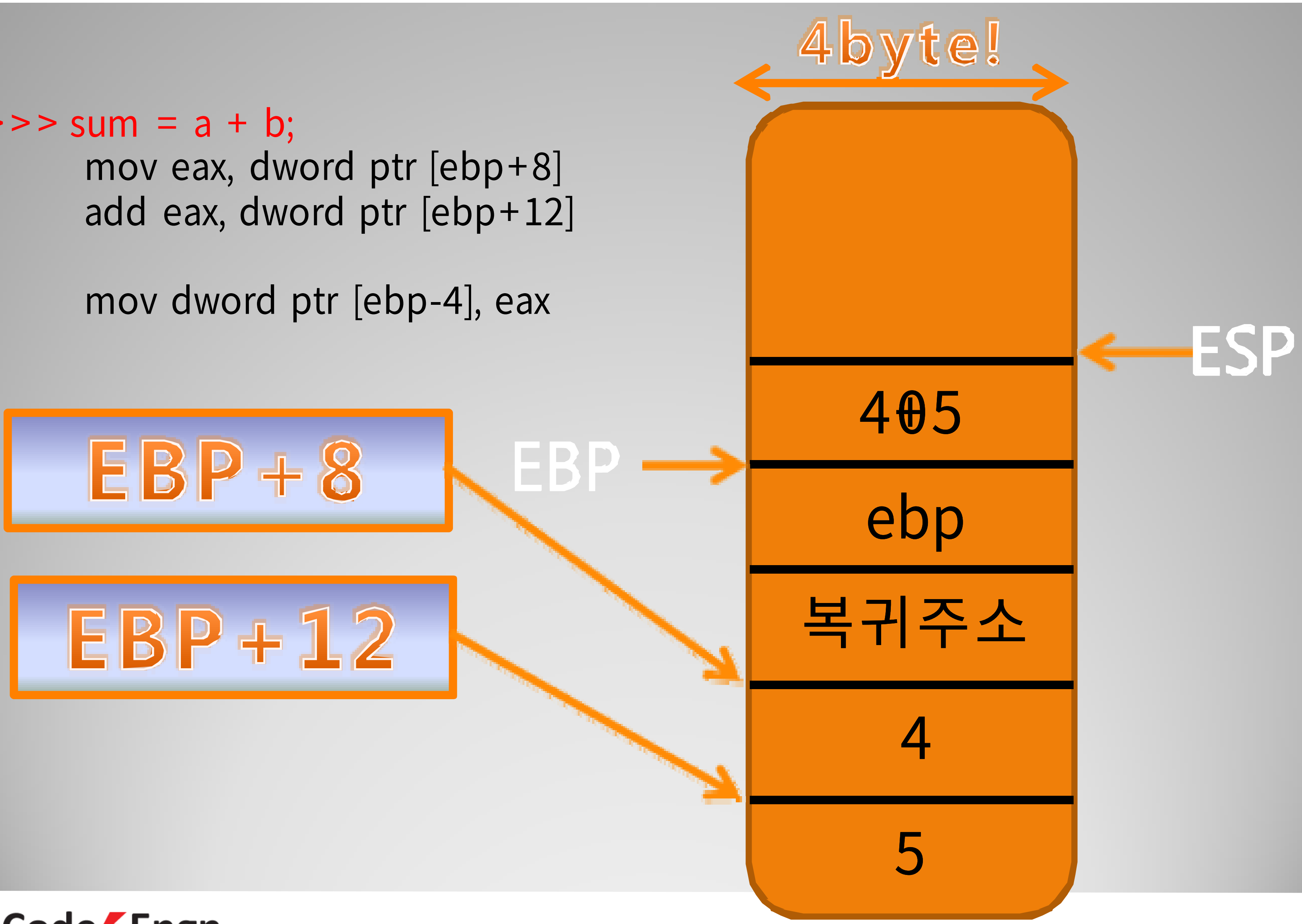
```
>>> int sum;
        sub esp, 4
```



```
>>> sum = 0;

      mov DWORD PTR [ebp-4], 0
```

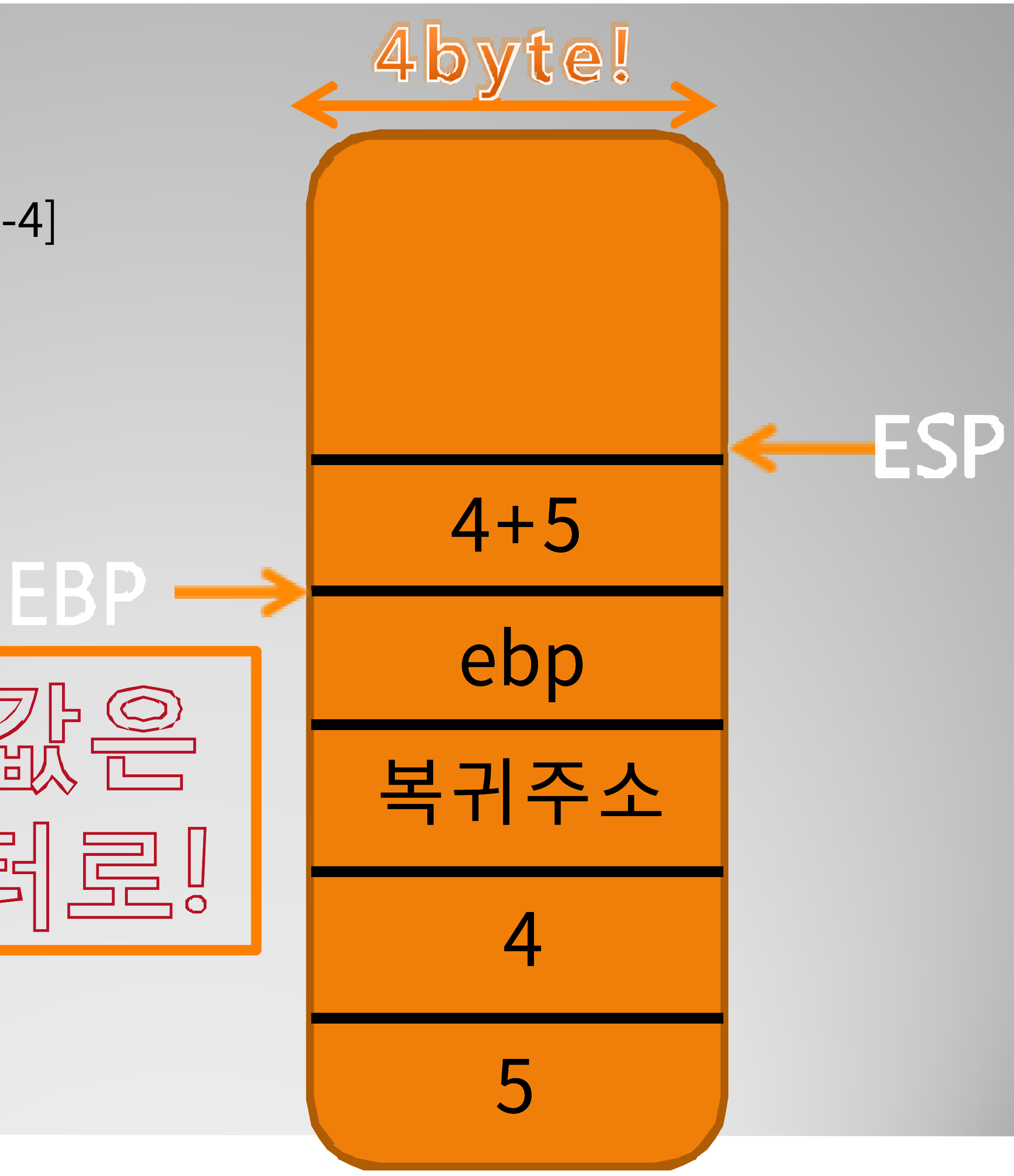


```
>>> sum = a + b;
        mov eax, dword ptr [ebp+8]
        add eax, dword ptr [ebp+12]

        mov dword ptr [ebp-4], eax
```

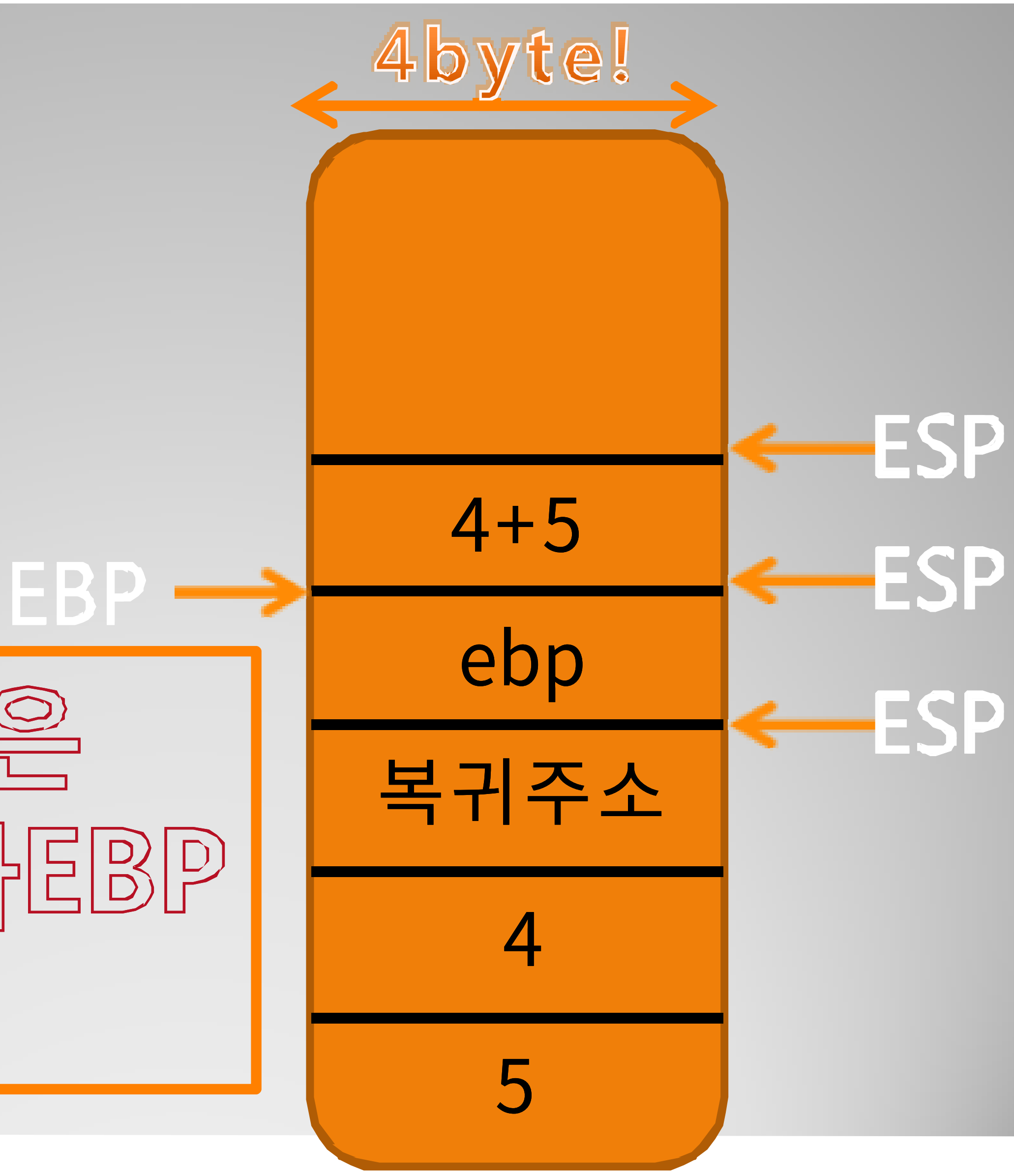


```
>>> return sum;
        mov eax, dword ptr [ebp-4]
```

함수의 반환값은
EAX 레지스터로!



```
mov esp, ebp
        pop ebp
```

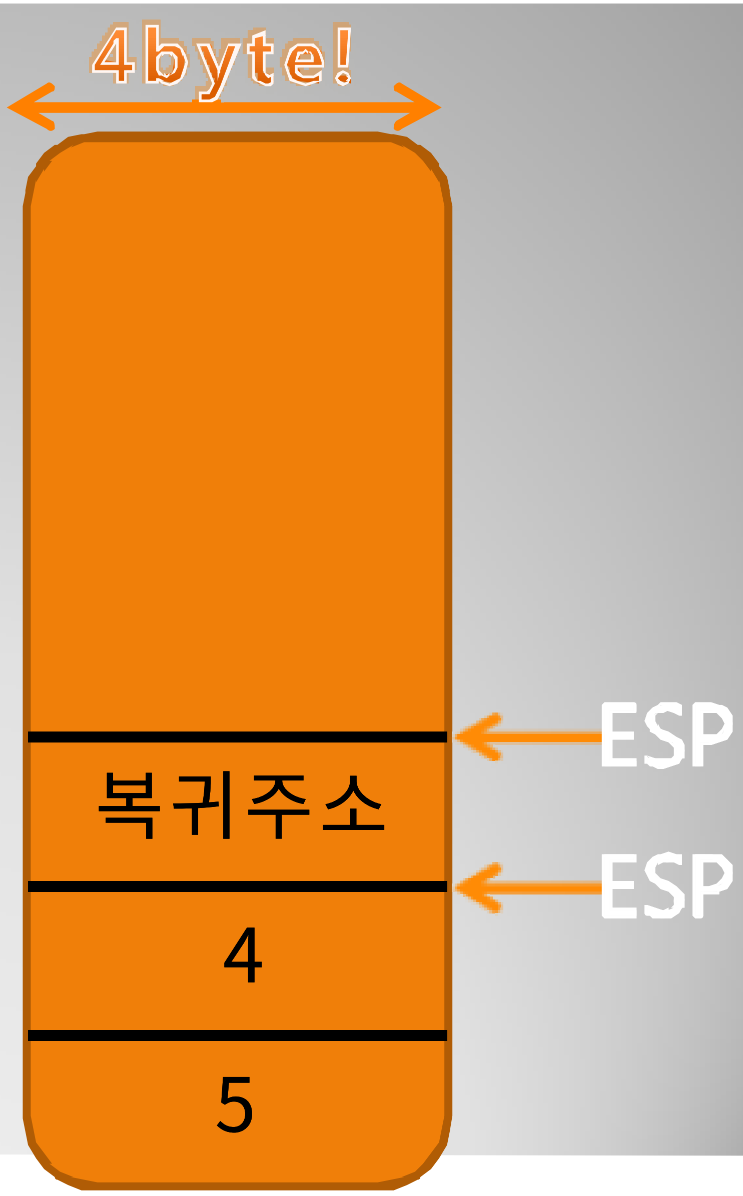


함수의 끝은 스택포인터와EBP 복구!

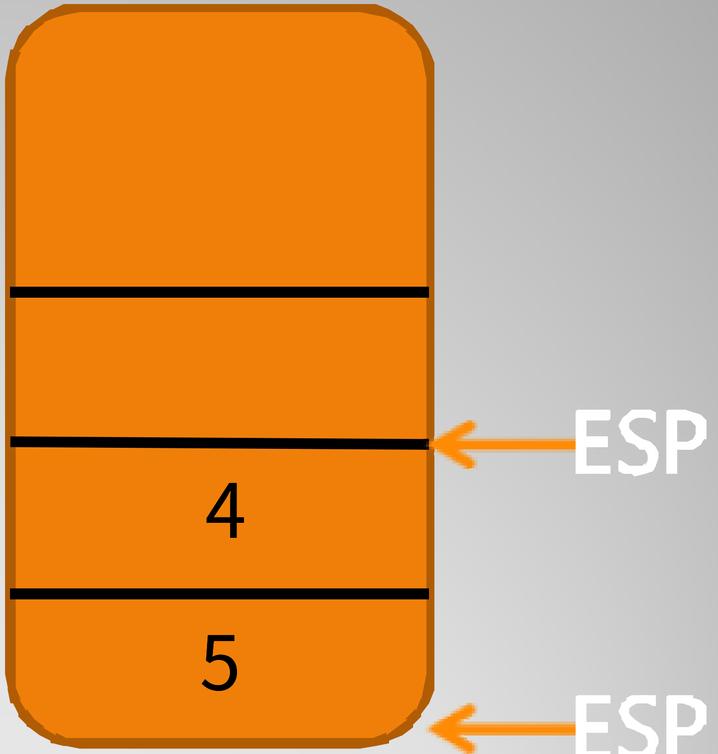
Code Engn
http://www.CodeEngn.com

ret

ret =
pop 복귀주소
jmp 복귀주소



```
push 5
push 4
call sigma
add esp, 8
```



空手來空手去

스택을 쓰는 이유 : 자료를 잠깐 보관!

스택의 장점 : 다른 자료형은 자료를 저장하고 지우면 상태가 변하지만 스택은 항상 이전 상태로 돌아온다

- C와 어셈블리 같이 쓰기
- 인라인 어셈블리는 컴파일러마다 문법이 다릅니다.
- [illegible]
- [illegible]

왠만하면 그냥 C로 하자!
정신 건강에 좋음!

- [illegible]
  - 하드웨어를 직접 제어할 때
  - 프로세서가 지원하는 특이한 명령어를 사용할 때 (MMX, SSE)

Calling conventions
(cont'd)

Code Engn
http://www.CodeEngn.com

- C 문법에서는 함수가 호출하기 전과 호출 후
- 모든 레지스터 값이 변하지 않는다고 가정합니다.
- 따라서 어셈블리 함수에서는
- PUSHA/POPA, PUSHF/POPF 명령어를 써서
- 모든 레지스터를 보존해야 합니다.
- 예외! EAX는 함수의 반환값을 저장하므로 보존하지 않아도 됩니다.

Calling conventions
(cont'd)

- 많은 C 컴파일러는 컴파일을 하면서 함수 이름에 _를 붙여서 어셈블리 코드를 만듭니다.
- GCC는 예외입니다.
- Visual Studio에서 디스어셈블 창을 열어서 확인해보세요.
- GCC –S 도 해보세요.

Calling conventions
(cont'd)

Calling conventions
(cont’d)

- 함수 인자 전달하기

- 뒤에 있는 인자부터 스택에 넣기
- C Calling Convention은 인자가 아무리 많아도 혹은 인자의 개수가 고정되지 않아도 상관없도록 설계됨
  - int printf(char *, …);
- 특히 printf 함수에서는 포맷 문자열이 ebp+8이고 나머지 인자가 아무리 많아도 상관없음
- 단, 포맷 문자열에 몇 개의 인자가 있는지 알 수 있도록 표시를 해야 함
  - printf(“x = %d \ n”); -> 1개의 값을 스택에서 읽어옴

Code⚡Engn
http://www.CodeEngn.com

- 변수의 주소 계산

- 전역 변수는 항상 데이터 세그먼트에 있음
- 지역 변수 (자동 변수)는 스택에 있음
- 따라서 전역 변수는 함수가 끝나도 계속 존재
- 지역 변수는 함수가 끝나면 사라짐
- 지역 변수는 항상 포인터 값을 이용해서 계산됨
  - mov eax, [ebp-8]
- 빠른 처리가 필요한 데이터는 전역 변수로!
- 전역 변수가 많으면 파일이 커지고 캐쉬 문제 발생
- 이게 바로 Trade-Off

Calling conventions (cont'd)

- 반환 값

- 정수 값일 경우에는 eax에 저장됨
- char나 short 형 변수도 32비트로 늘려짐
- 부동소수점 변수는 st0 레지스터에 저장됨

..왜 반환 값은 한 개만 될까요?
..포인터 형은 어떻게 될까요?
..숙제에요~

Calling conventions
(cont'd)

Code Engn
http://www.CodeEngn.com

NASM
(Netwide Assembler)

뭔말인지는 모름

Code Engn
http://www.CodeEngn.com

- 자세한 설명은 아래 링크로!!

'PC ASSEMBLY LANGUAGE'
BY PAUL A. CARTER
HTTP://WWW.DRPAULCARTER.COM/

- 수업 교재라서 쉽고 자세하게 설명해줍니다.
- 저도 이 문서로 공부했어요.
- 돈 주고 책 사지 마세요

NASM

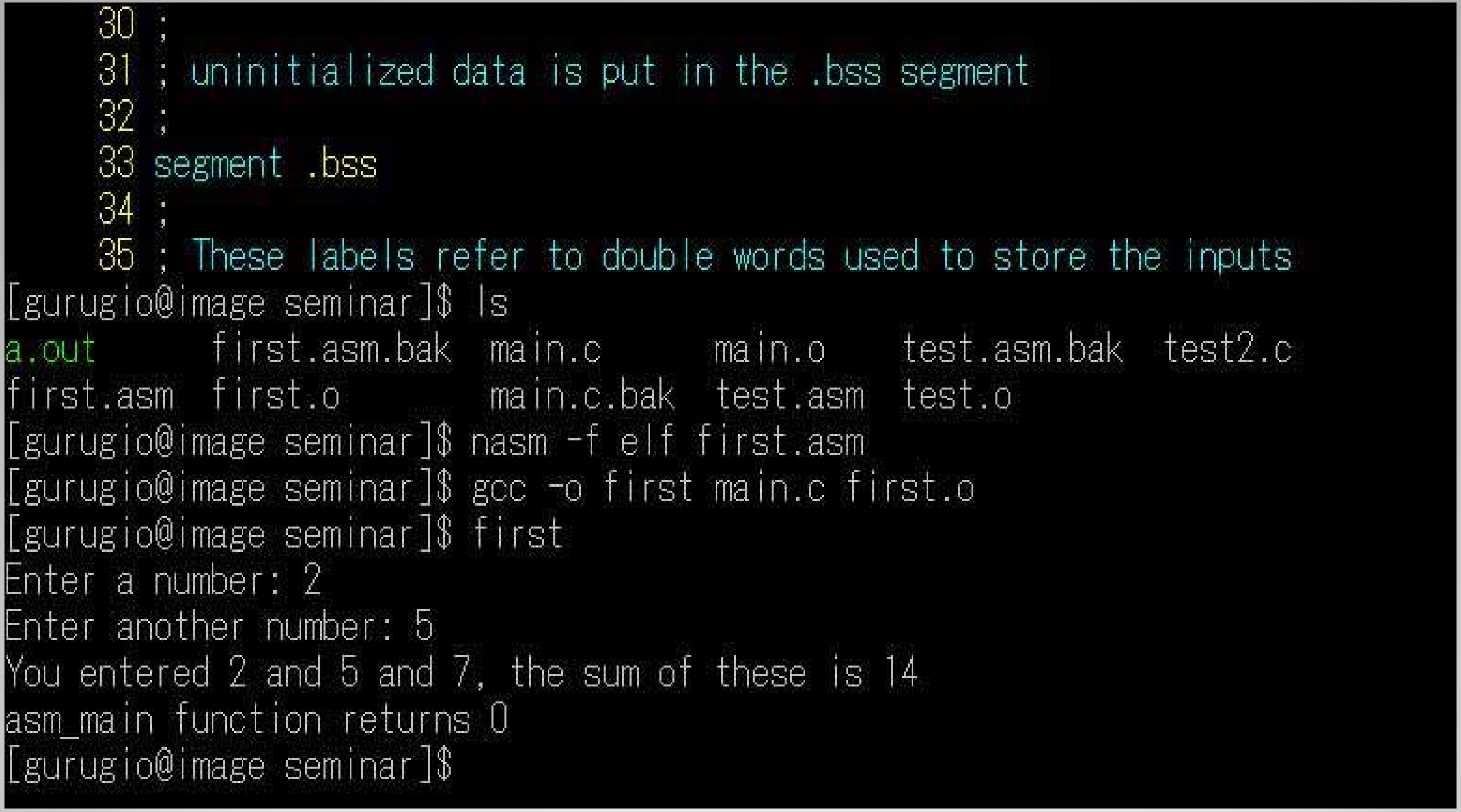
- NASM 다운받기

- http://nasm.sourceforge.net/
- Windows와 Linux, BSD 등등 여러 플랫폼에서 사용할 수 있습니다.
- 인텔 표준 문법 (MASM) 과 비슷하면서도 더 쉽고 간단합니다.
- gcc와 같이 사용하면 더 좋아요.
- MMX, SSE2, 3DNOW! 등 최신 프로세서의 명령어를 지원합니다.

..라고 매뉴얼에 써있습니다.

NASM

사용법



```
;
; uninitialized data is put in the .bss segment
;
segment .bss
;
; These labels refer to double words used to store the inputs
[gurugio@image seminar]$ ls
a.out       first.asm.bak  main.c      main.o    test.asm.bak  test2.c
first.asm   first.o        main.c.bak  test.asm  test.o
[gurugio@image seminar]$ nasm -f elf first.asm
[gurugio@image seminar]$ gcc -o first main.c first.o
[gurugio@image seminar]$ first
Enter a number: 2
Enter another number: 5
You entered 2 and 5 and 7, the sum of these is 14
asm_main function returns 0
[gurugio@image seminar]$
```

오브젝트 파일을 만든 후 GCC로 링크하세염~

```
L1     db              0           ; byte
L2     dw              1000        ; word
L3     db              110101b     ; byte
L4     db              12h         ; byte
L5     db              17o         ; byte
L6     dd              1A92h ; double word
L7     resb            1           ; uninitialized byte
L8     db              'A'         ; ascii code = 'A'
L9     db              0,1,2,3 ; 4 bytes
L10    db              'w', 'o','r','d',0 ;string
L11    db              'word', 0
L12    times 100 db 0        ; 100 bytes of zero
L13    resw            100   ; 100*2(word bytes
```

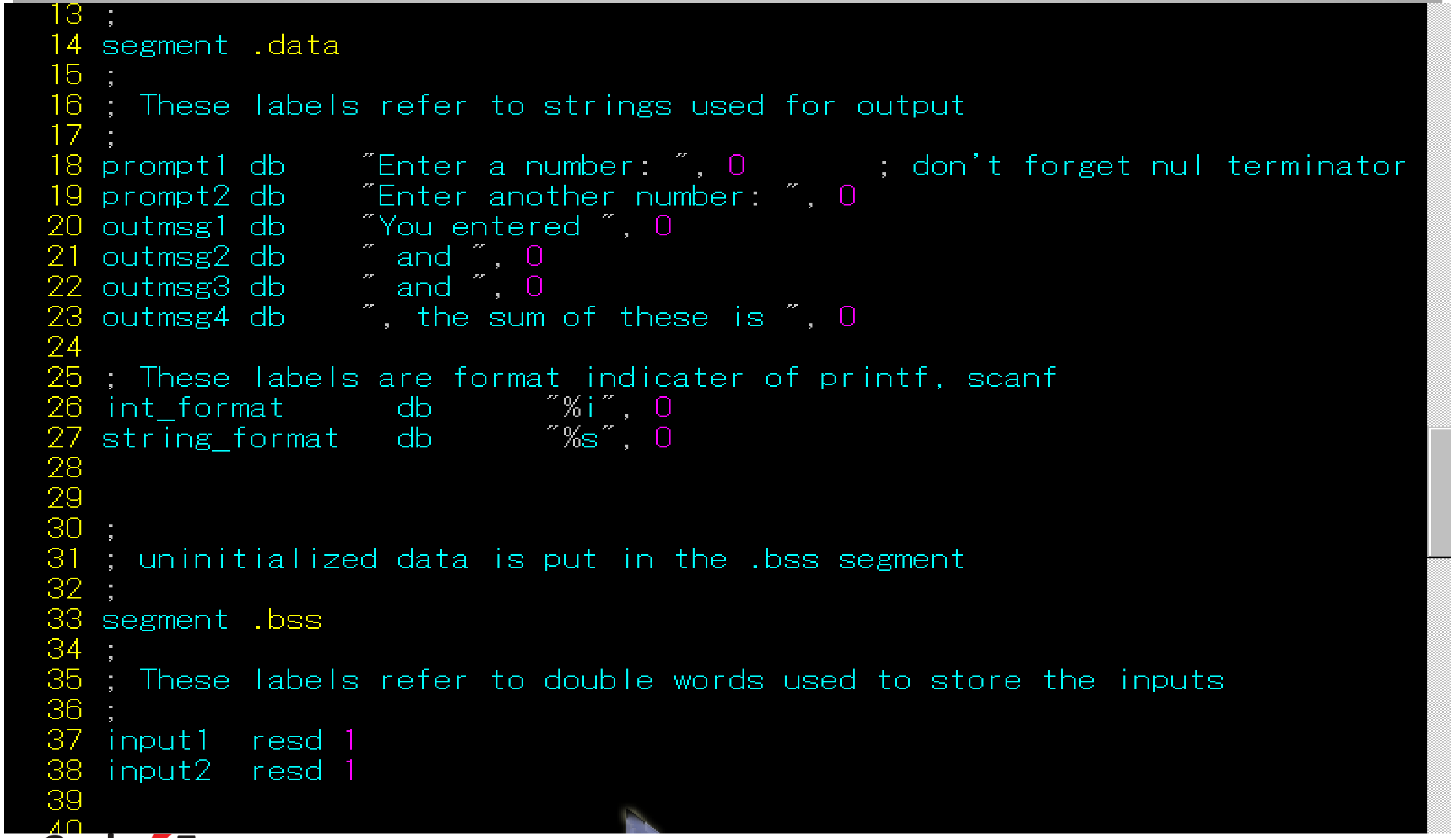
NASM

- Mov al, [L1] ;copy byte at L1
- Mov eax, L1   ;eax = address of byte at L1
- Mov [L1], ah ; copy ah into byte at L1
- Mov eax, [L6] ; copy double word
- Add eax, [L6] ; eax = eax + double word at L6
- Add [L6], eax ; double word at L6 += eax
- Mov al, [L6]; copy first byte of double word at L6 into al
- Mov [L6], 1   ; operation size is not specified
- Mov dword [L6], 1 ; store a 1 at L6

영어 공부 합시당~

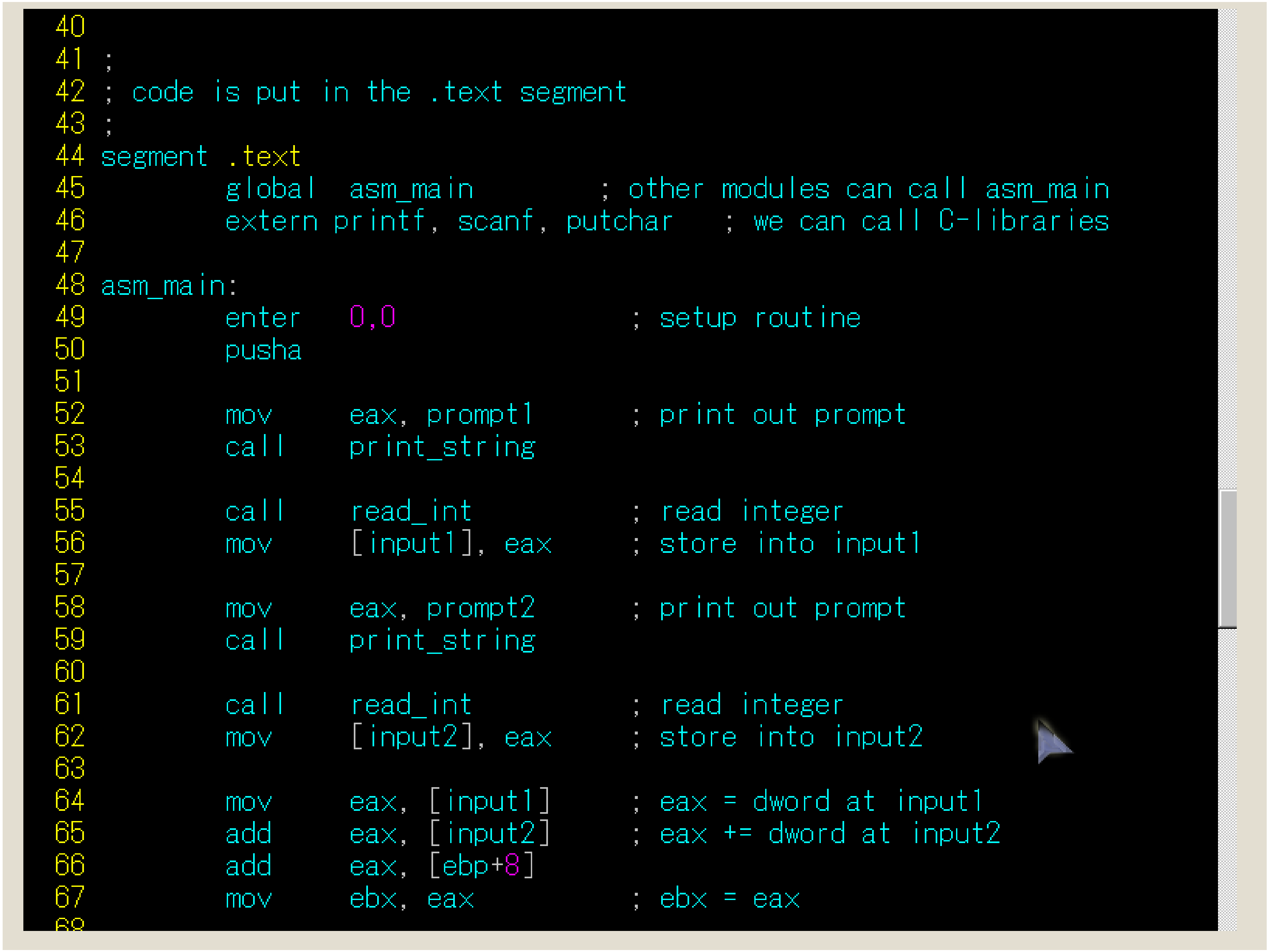
NASM

한번 읽어나 봅시다

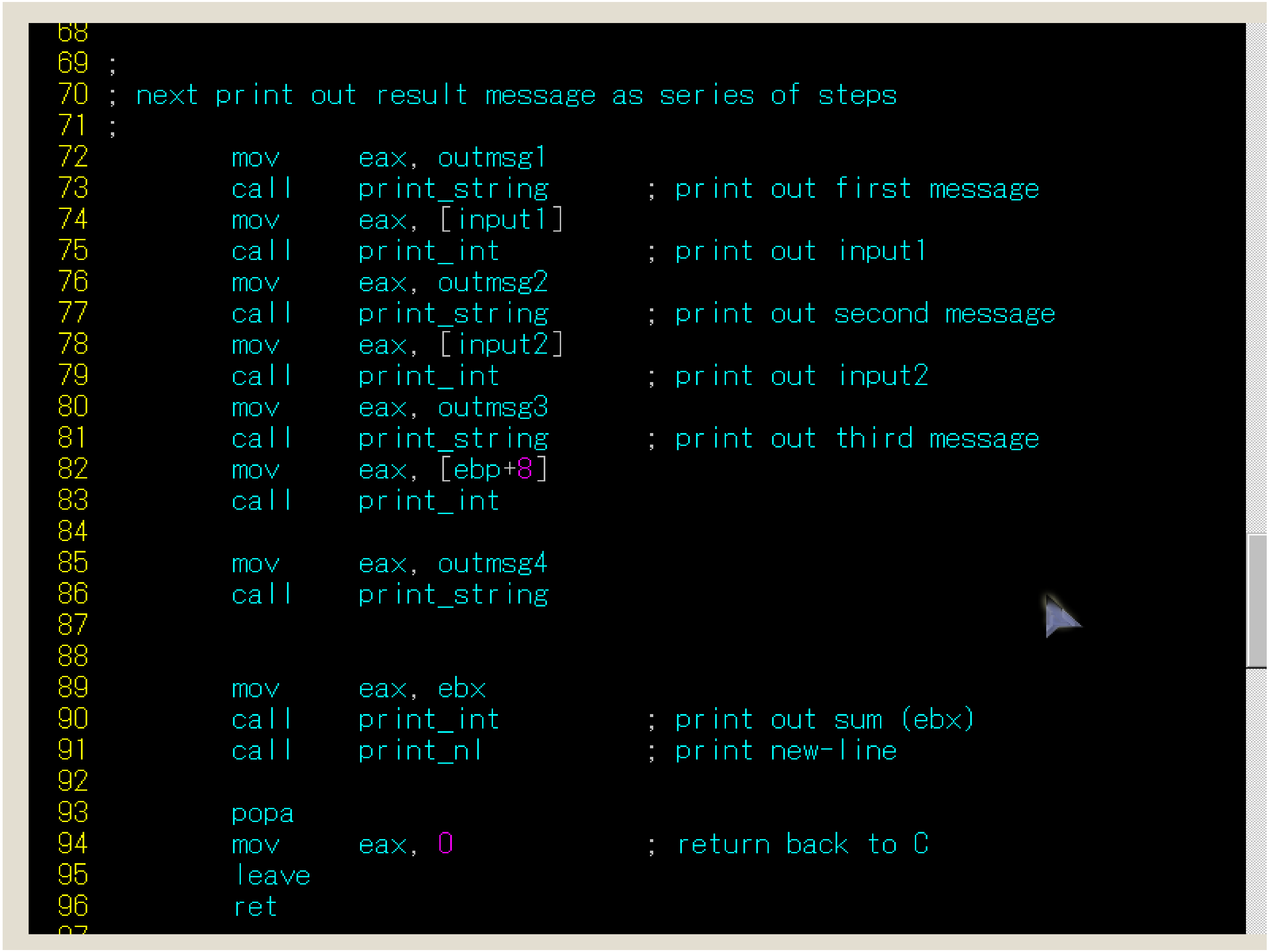


```
;
segment .data
;
; These labels refer to strings used for output
;
prompt1 db    "Enter a number: ", 0       ; don't forget nul terminator
prompt2 db    "Enter another number: ", 0
outmsg1 db    "You entered ", 0
outmsg2 db    " and ", 0
outmsg3 db    " and ", 0
outmsg4 db    ", the sum of these is ", 0

; These labels are format indicater of printf, scanf
 int_format       db       "%i", 0
string_format    db       "%s", 0



;
; uninitialized data is put in the .bss segment
;
segment .bss
;
; These labels refer to double words used to store the inputs
;
 input1   resd 1
 input2   resd 1

```

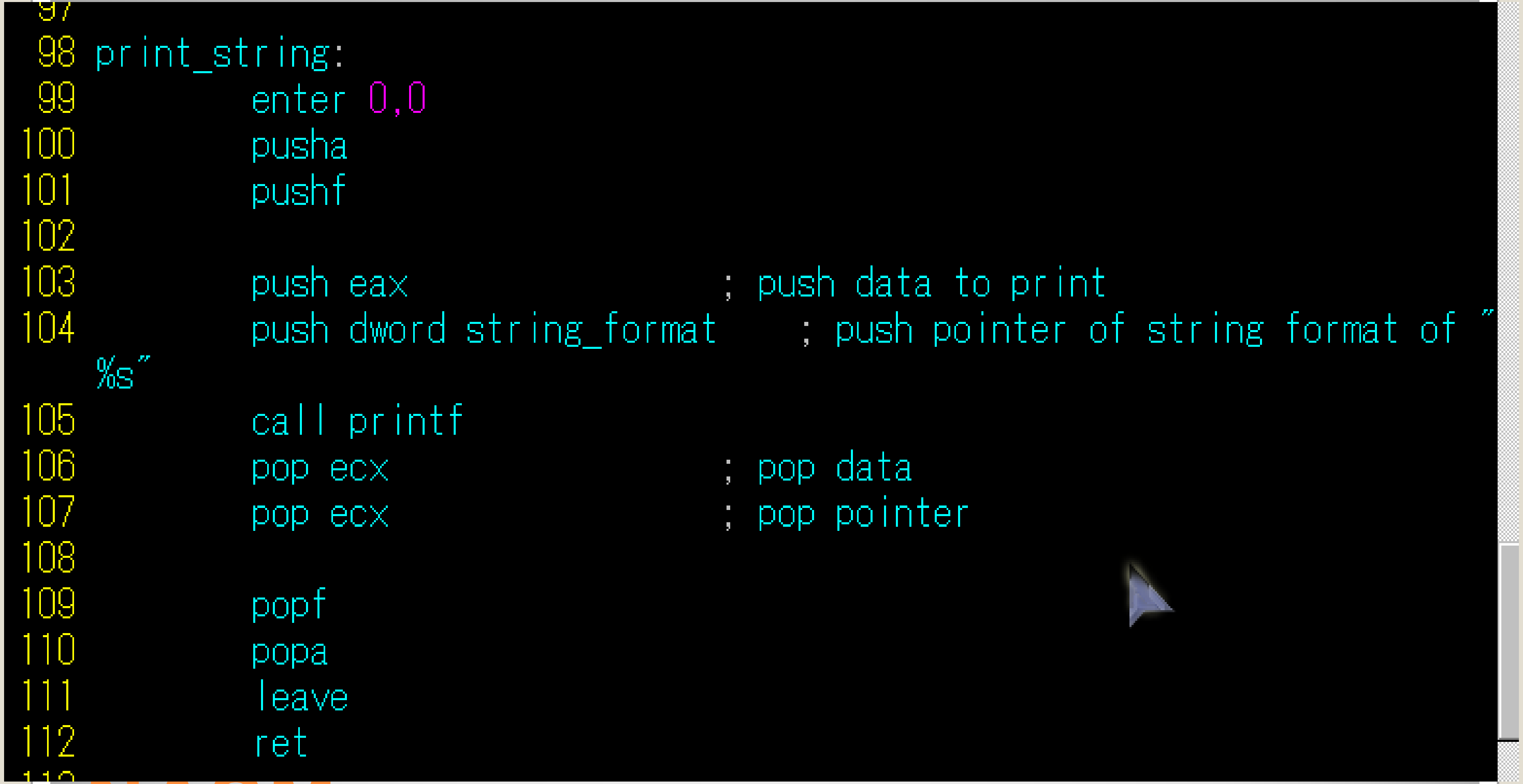
Code Engn
http://www.CodeEngn.com



```
;
; code is put in the .text segment
;
segment .text
        global  asm_main          ; other modules can call asm_main
        extern printf, scanf, putchar   ; we can call C-libraries

asm_main:
        enter   0,0               ; setup routine
        pusha

        mov     eax, prompt1      ; print out prompt
        call    print_string

        call    read_int          ; read integer
        mov     [input1], eax     ; store into input1

        mov     eax, prompt2      ; print out prompt
        call    print_string

        call    read_int          ; read integer
        mov     [input2], eax     ; store into input2

        mov     eax, [input1]     ; eax = dword at input1
        add     eax, [input2]     ; eax += dword at input2
        add     eax, [ebp+8]
        mov     ebx, eax          ; ebx = eax
```



```
;
; next print out result message as series of steps
;
        mov     eax, outmsg1
        call    print_string        ; print out first message
        mov     eax, [input1]
        call    print_int           ; print out input1
        mov     eax, outmsg2
        call    print_string        ; print out second message
        mov     eax, [input2]
        call    print_int           ; print out input2
        mov     eax, outmsg3
        call    print_string        ; print out third message
        mov     eax, [ebp+8]
        call    print_int

        mov     eax, outmsg4
        call    print_string


        mov     eax, ebx
        call    print_int           ; print out sum (ebx)
        call    print_nl            ; print new-line

        popa
        mov     eax, 0              ; return back to C
        leave
        ret
```

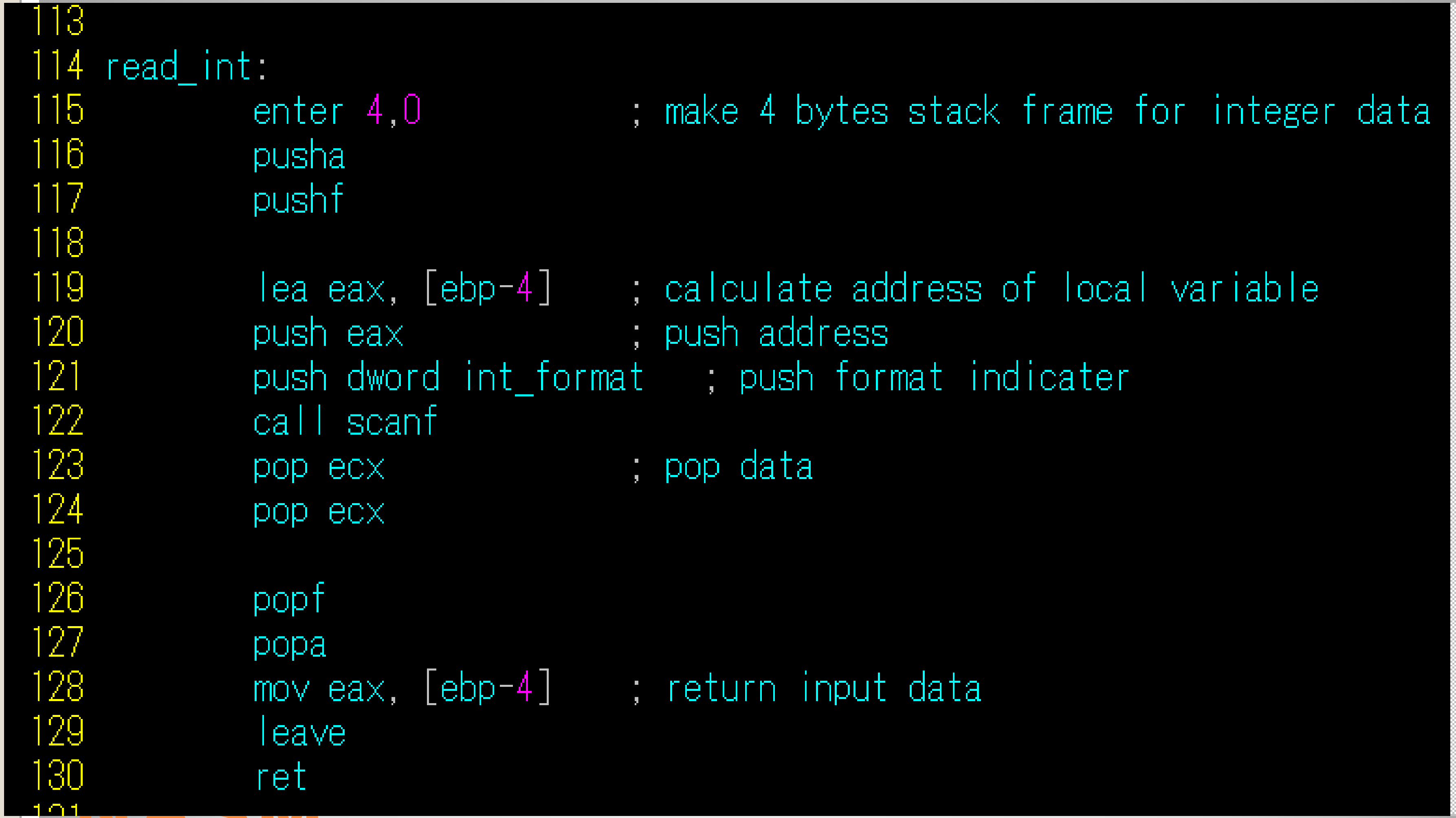


```
print_string:
        enter 0,0
        pusha
        pushf

        push eax                ; push data to print
        push dword string_format    ; push pointer of string format of "%s"
        call printf
        pop ecx                 ; pop data
        pop ecx                 ; pop pointer

        popf
        popa
        leave
        ret
```

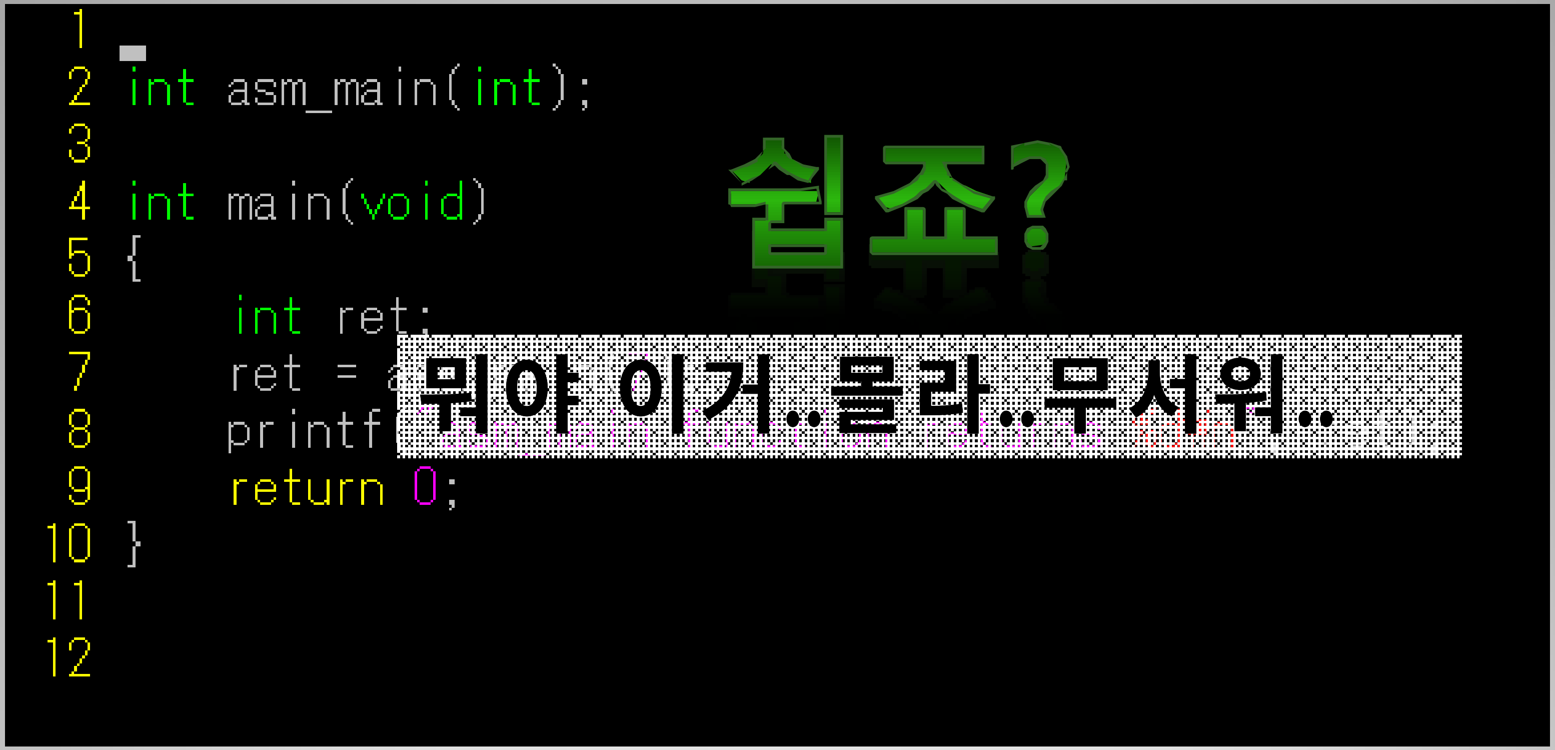
NASM

Code⚡Engn
http://www.CodeEngn.com

NASM



```
read_int:
        enter 4,0               ; make 4 bytes stack frame for integer data
        pusha
        pushf

        lea eax, [ebp-4]        ; calculate address of local variable
        push eax                ; push address
        push dword int_format       ; push format indicater
        call scanf
        pop ecx                 ; pop data
        pop ecx

        popf
        popa
        mov eax, [ebp-4]        ; return input data
        leave
        ret
```



```

int asm_main(int);

int main(void)
{
        int ret;
        ret = asm_main(0);
        printf("asm_main function returns %d\n", ret);
        return 0;
}


```

NASM

마 무 리

- 재밌었나요?
- 흥분되나요?
- 재밌었으면 당신은 해커~
- 흥분되었으면 당신은 오덕후~

눈뜨세요~

- IT 업계의 현실을 아시나요?
- 점점 좋아지고 있다고들 합니다.
- 그래도 아직은 많이 어둡지요.
- 해뜨기 직전 제일 어둡지요.
- 우리 모두가 노력하면 이겨낼 수 있습니다.
- 환경만 탓하지 말고 우리 실력을 키워요

- 함께 해주셔서 감사합니다.
- 썰렁한 농담 죄송합니다.
- 지금까지 고냥이었습니다.
- 예쁘게 봐주세염~

화장실 갑시다~ 커피도 한잔~