Introduction to 8086 Assembly

Lecture 13

Inline Assembly

Inline Assembly

- Compiler-dependent
- GCC -> GAS (the GNU assembler)

Intel Syntax => AT&T Syntax

- Registers: `eax` => `%eax`
- Immediates: `123` => `$123`
- Memory:
  - `lbl1` => `$lbl1` (address of lbl1)
  - `[lbl1]` => `lbl1` (content of lbl1)

Intel Syntax => AT&T Syntax

- Operand order reversed:
  - `mov dest, src  =>  mov src, dest`
- Operand size in command (`movb`, `movw`, `movl`, `addb`, `addw`, `addl`, etc):
  - `mov eax, ebx  => movl %ebx, %eax`
  - `add dl, ch    => addb %ch, %dl`
- Indirect addressing
  - `mov eax, [ebx]   => movl (%ebx), %eax`
  - `add ax, [ebx+4] => addw 4(%ebx), %ax`
  - `mov dword [ebx], 1 => movl $1, (%ebx)`

Compile C to AT&T Assembly

- `gcc -S  myprogram.c`
- `gcc -S  -masm=att   myprogram.c`

More on Intel vs. AT&T Syntax

- https://en.wikipedia.org/wiki/X86_assembly_language#Syntax
- https://en.wikibooks.org/wiki/X86_Assembly/GAS_Syntax
- https://imada.sdu.dk/Courses/DM18/Litteratur/IntelnATT.htm

Basic inline assembly

inline1.c

```c
int main() {

  asm ("movl $1, %eax");

  return 0;
}
```

inline2.c

```c
int main() {

  __asm__("movl %eax, %ebx");

  return 0;
}
```

Basic inline assembly

inline3.c

```c
int main() {
  int a;

  asm("movl $10, %eax; xchg %al, %ah");

  asm("movl $10, %eax;"
          "xchg %al, %ah");

  return 0;
}
```

Global symbols (functions, global variables)

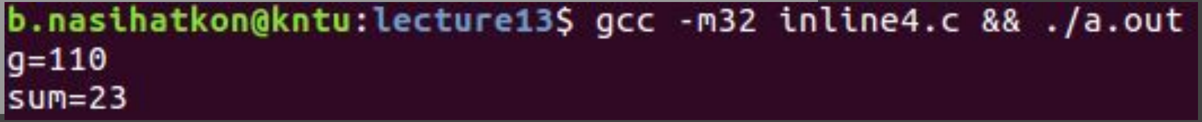
inline4.c

```
#include <stdio.h>

int g = 0;

void print_sum(int a, int b) {
  printf("sum=%d\n",a+b);
}

int main() {

  asm ("movl $110, g");   // NASM: mov dword [g], 110

  printf("g=%d\n",g);

  asm ("pushl $10;"       // NASM: push 10
       "pushl $13;"       // NASM: push 13
       "call print_sum;"  // NASM: call print_sum
       "addl $8, %esp;"); // NASM: add esp, 8

  return 0;
}
```

Global symbols (functions, global variables)

inline4.c

```
#include <stdio.h>

int g = 0;

void print_sum(int a, int b) {
  printf("sum=%d\n",a+b);
}

int main() {

  asm ("movl $110, g");   // NASM: mov dword [g], 110

  printf("g=%d\n",g);

  asm ("pushl $10;"        // NASM: push 10
       "pushl $13;"        // NASM: push 13
       "call print_sum;"   // NASM: call print_sum
       "addl $8, %esp;"); // NASM: add esp, 8

  return 0;
}
```



```
b.nasihatkon@kntu:lecture13$ gcc -m32 inline4.c && ./a.out
g=110
sum=23
```

Global symbols (functions, global variables)

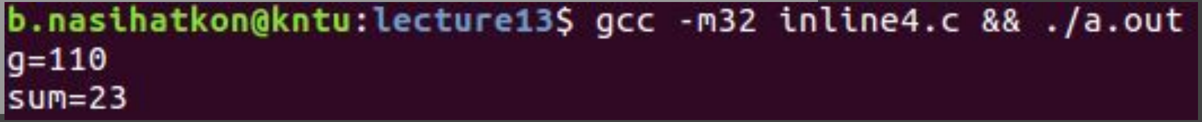
K. N. Toosi
University of Technology

inline4.c

```
#include <stdio.h>

int g = 0;

void print_sum(int a, int b) {
  printf("sum=%d\n",a+b);
}
```

Do not use this technique!
It might not alway work!

```
  printf("g=%d\n",g);

  asm ("pushl $10;"         // NASM: push 10
       "pushl $13;"         // NASM: push 13
       "call print_sum;"    // NASM: call print_sum
       "addl $8, %esp;");  // NASM: add esp, 8

  return 0;
}
```



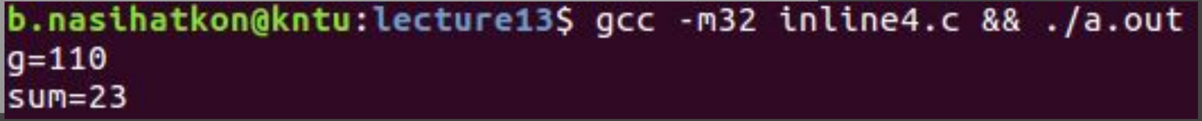
```
b.nasihatkon@kntu:lecture13$ gcc -m32 inline4.c && ./a.out
g=110
sum=23
```

Global symbols (functions, global variables)

inline4.c

```
#include <stdio.h>

int g = 0;

void print_sum(int a, int b) {
  printf("sum=%d\n",a+b);
}

int mai

  asm ('

  printf("g=%d\n",g);

  asm ("pushl $10;"         // NASM: push 10
       "pushl $13;"         // NASM: push 13
       "call print_sum;"    // NASM: call print_sum
       "addl $8, %esp;");   // NASM: add esp, 8

  return 0;
}
```

what about local variables?



```
b.nasihatkon@kntu:lecture13$ gcc -m32 inline4.c && ./a.out
g=110
sum=23
```

How GCC handles inline assembly?

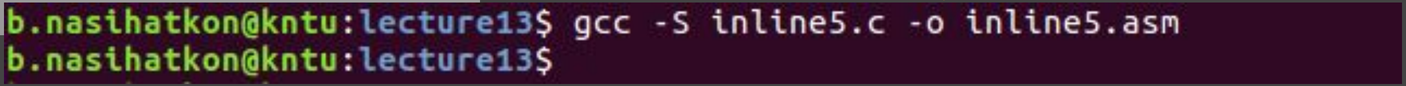
inline5.c

```
#include <stdio.h>

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

  scanf("%d %d", &a, &b);

  c = a+b;

  asm ("charand_command %ebx, %eax");

  printf("a=%d b=%d, a+b=%d\n", a, b, c);

  return 0;
}
```

How GCC handles inline assembly?

inline5.c

```c
#include <stdio.h>

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

  scanf("%d %d", &a, &b);

  c = a+b;

  asm ("charand_command %ebx, %eax");

  printf("a=%d b=%d, a+b=%d\n", a, b, c);

  return 0;
}
```



```
b.nasihatkon@kntu:lecture13$ gcc -S inline5.c -o inline5.asm
b.nasihatkon@kntu:lecture13$
```

No Error Compiling to Assembly!

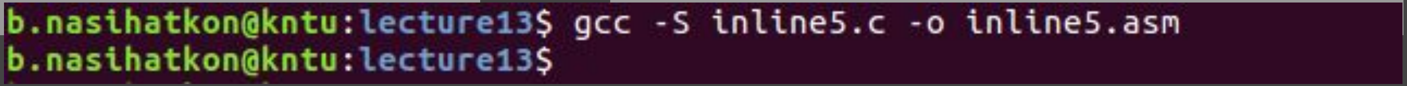
How GCC handles inline assembly?

inline5.c

```c
#include <stdio.h>

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

  scanf("%d %d", &a, &b);

  c = a+b;

  asm ("charand_command %ebx, %eax");

  printf("a=%d b=%d, a+b=%d\n", a, b, c);

  return 0;
}
```

inline5.asm

```
            :
    movl   -16(%rbp), %eax
    addl   %edx, %eax
    movl   %eax, -12(%rbp)

    charand_command %ebx, %eax

    movl   -16(%rbp), %edx
    movl   -20(%rbp), %eax
    movl   -12(%rbp), %ecx
    movl   %eax, %esi
    movl   $.LC1, %edi
    movl   $0, %eax
            :
```



```
b.nasihatkon@kntu:lecture13$ gcc -S inline5.c -o inline5.asm
b.nasihatkon@kntu:lecture13$
```

How GCC handles inline assembly?

K. N. Toosi
University of Technology

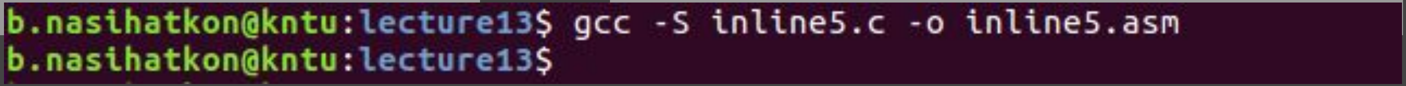
inline5.c

```c
#include <stdio.h>

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

  scanf("%d %d", &a, &b);

  c = a+b;

  asm ("charand_command %ebx, %eax");

  printf("a=%d b=%d, a+b=%d\n", a, b, c);

  return 0;
}
```

inline5.asm

```
          :
    movl  -16(%rbp), %eax
    addl  %edx, %eax
    movl  %eax, -12(%rbp)

    charand_command %ebx, %eax

    movl  -16(%rbp), %edx
    movl  -20(%rbp), %eax
    movl  -12(%rbp), %ecx
    movl  %eax, %esi
    movl  $.LC1, %edi
    movl  $0, %eax
          :
```

just inserting inline assembly



```
b.nasihatkon@kntu:lecture13$ gcc -S inline5.c -o inline5.asm
b.nasihatkon@kntu:lecture13$
```

How GCC handles inline assembly?

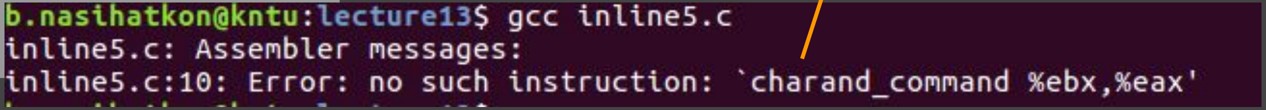
K. N. Toosi
University of Technology

inline5.c

```c
#include <stdio.h>

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

  scanf("%d %d", &a, &b);

  c = a+b;

  asm ("charand_command %ebx, %eax");

  printf("a=%d b=%d, a+b=%d\n", a, b, c);

  return 0;
}
```

GCC just inserts inline assembly!

Assembler Error!



```
b.nasihatkon@kntu:lecture13$ gcc inline5.c
inline5.c: Assembler messages:
inline5.c:10: Error: no such instruction: `charand_command %ebx,%eax'
```

How GCC handles inline assembly?

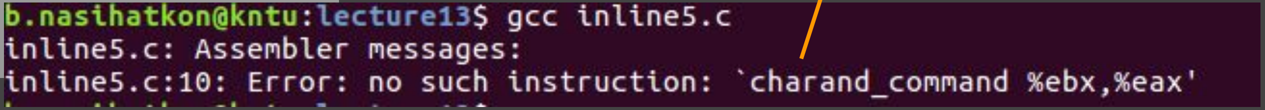
K. N. Toosi
University of Technology

inline5.c

```
#include <stdio.h>

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

  scanf("%d %d", &a, &b);

  c = a+b;

  asm ("charand_command %ebx, %eax");

  printf("a=%d b=%d, a+b=%d\n", a, b, c);

  return 0;
}
```

GCC just inserts inline assembly!
It has no idea what inline assembly is doing!

Assembler Error!



```
b.nasihatkon@kntu:lecture13$ gcc inline5.c
inline5.c: Assembler messages:
inline5.c:10: Error: no such instruction: `charand_command %ebx,%eax'
```

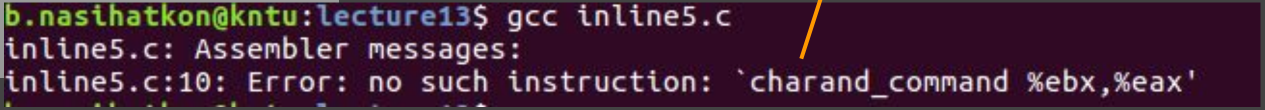
How GCC handles inline assembly?

inline5.c

```
#include <stdio.h>

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

  scanf("%d %d", &a, &b);

  c = a+b;

  asm ("charand_command %ebx, %eax");

  printf("a=%d b=%d, a+b=%d\n", a, b, c);

  return 0;
}
```

GCC just inserts inline assembly!
It has no idea what inline assembly is doing!
=> side effects!

Assembler Error!



```
b.nasihatkon@kntu:lecture13$ gcc inline5.c
inline5.c: Assembler messages:
inline5.c:10: Error: no such instruction: `charand_command %ebx,%eax'
```

What can go wrong?

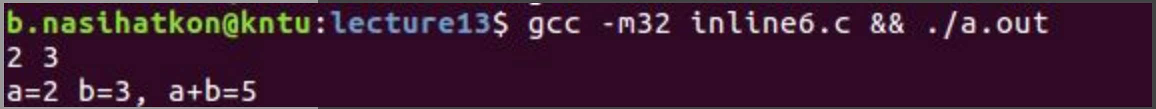
inline6.c

```
#include <stdio.h>

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

  scanf("%d %d", &a, &b);

  c = a+b;

  asm ("movl $1, %eax;"
          "movl $1, %ebx;"
          "movl $1, %ecx;"
          "movl $1, %edx");

  printf("a=%d b=%d, a+b=%d\n", a, b, c);

  return 0;
}
```

What can go wrong?

inline6.c

```c
#include <stdio.h>

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

  scanf("%d %d", &a, &b);

  c = a+b;

  asm ("movl $1, %eax;"
       "movl $1, %ebx;"
       "movl $1, %ecx;"
       "movl $1, %edx");

  printf("a=%d b=%d, a+b=%d\n", a, b, c);

  return 0;
}
```



```
b.nasihatkon@kntu:lecture13$ gcc -m32 inline6.c && ./a.out
2 3
a=2 b=3, a+b=5
```

What can go wrong? Case 1:

inline7.c

```
#include <stdio.h>

int main() {
  int a,b;

  register int c;

  scanf("%d %d", &a, &b);

  c = a+b;

  asm ("movl $1, %eax;"
          "movl $1, %ebx;"
          "movl $1, %ecx;"
          "movl $1, %edx");

  printf("a=%d b=%d, a+b=%d\n", a, b, c);

  return 0;
}
```

What can go wrong? Case 1:

inline7.c

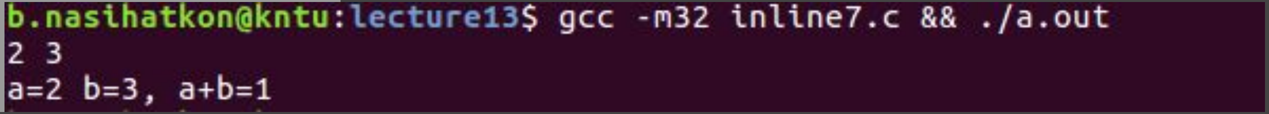
```
#include <stdio.h>

int main() {
  int a,b;

  register int c;

  scanf("%d %d", &a, &b);

  c = a+b;

  asm ("movl $1, %eax;"
         "movl $1, %ebx;"
         "movl $1, %ecx;"
         "movl $1, %edx");

  printf("a=%d b=%d, a+b=%d\n", a, b, c);

  return 0;
}
```

gcc tries to use a register to store c

What can go wrong? Case 1:

inline7.c

```c
#include <stdio.h>

int main() {
  int a,b;

  register int c;

  scanf("%d %d", &a, &b);

  c = a+b;

  asm ("movl $1, %eax;"
         "movl $1, %ebx;"
         "movl $1, %ecx;"
         "movl $1, %edx");


  printf("a=%d b=%d, a+b=%d\n", a, b, c);

  return 0;
}
```



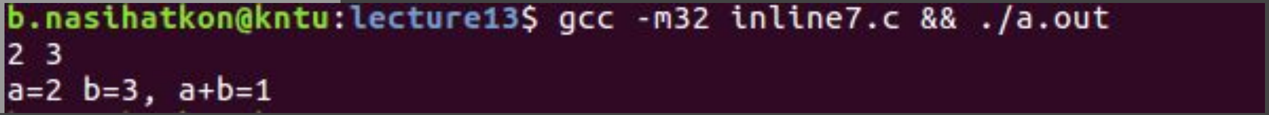
```
b.nasihatkon@kntu:lecture13$ gcc -m32 inline7.c && ./a.out
2 3
a=2 b=3, a+b=1
```

What can go wrong? Case 1:

inline7.c

```c
#include <stdio.h>

int main() {
  int a,b;

  register int c;

  scanf("%d %d", &a, &b);

  c = a+b;

  asm ("movl $1, %eax;"
          "movl $1, %ebx;"
          "movl $1, %ecx;"
          "movl $1, %edx");

  printf("a=%d b=%d, a+b=%d\n", a, b, c);

  return 0;
}
```

might or might not work as registers unexpectedly change. (worked in this case).

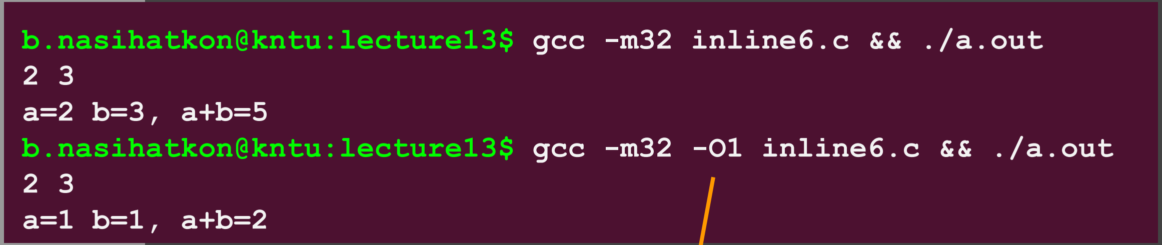


```
b.nasihatkon@kntu:lecture13$ gcc -m32 inline7.c && ./a.out
2 3
a=2 b=3, a+b=1
```

What can go wrong? Case 2:

inline6.c

```c
#include <stdio.h>

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

  scanf("%d %d", &a, &b);

  c = a+b;

  asm ("movl $1, %eax;"
       "movl $1, %ebx;"
       "movl $1, %ecx;"
       "movl $1, %edx");

  printf("a=%d b=%d, a+b=%d\n", a, b, c);

  return 0;
}
```



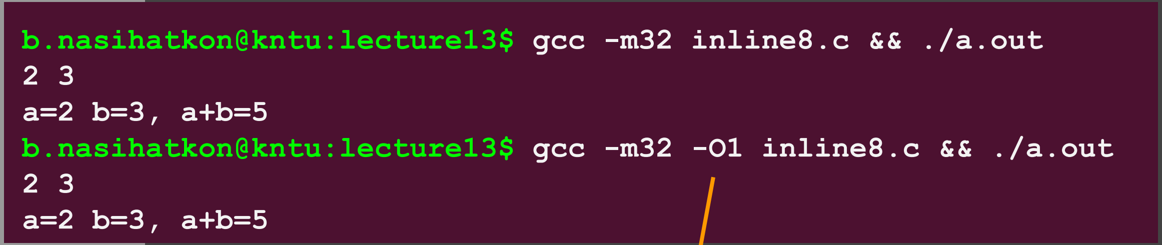
```
b.nasihatkon@kntu:lecture13$ gcc -m32 inline6.c && ./a.out
2 3
a=2 b=3, a+b=5
b.nasihatkon@kntu:lecture13$ gcc -m32 -O1 inline6.c && ./a.out
2 3
a=1 b=1, a+b=2
```

turn on optimization

Solution 1: use volatile keyword

inline8.c

```c
#include <stdio.h>

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

  scanf("%d %d", &a, &b);

  c = a+b;

  asm ("movl $1, %eax;"
       "movl $1, %ebx;"
       "movl $1, %ecx;"
       "movl $1, %edx");

  printf("a=%d b=%d, a+b=%d\n", a, b, c);

  return 0;
}
```



```
b.nasihatkon@kntu:lecture13$ gcc -m32 inline8.c && ./a.out
2 3
a=2 b=3, a+b=5
b.nasihatkon@kntu:lecture13$ gcc -m32 -O1 inline8.c && ./a.out
2 3
a=2 b=3, a+b=5
```

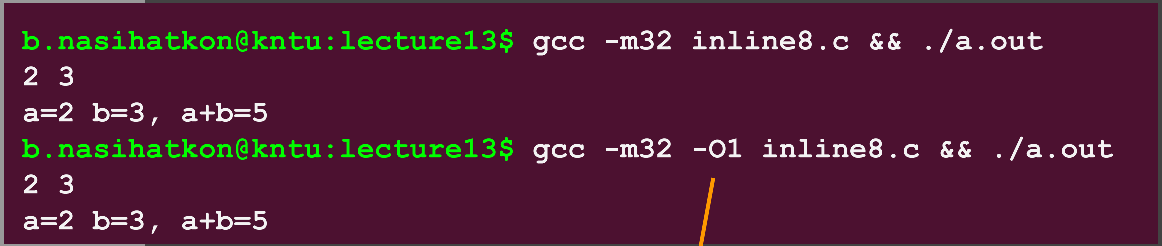
turn on optimization

Solution 1: use volatile keyword

inline8.c

```c
#include <stdio.h>

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

  scanf("%d %d", &a, &b);

  c = a+b;

  asm ("movl $1, %eax;"
       "movl $1, %ebx;"
       "movl $1, %ecx;"
       "movl $1, %edx");

  printf("a=%d b=%d, a+b=%d\n", a, b, c);

  return 0;
}
```

renders optimization useless!



```
b.nasihatkon@kntu:lecture13$ gcc -m32 inline8.c && ./a.out
2 3
a=2 b=3, a+b=5
b.nasihatkon@kntu:lecture13$ gcc -m32 -O1 inline8.c && ./a.out
2 3
a=2 b=3, a+b=5
```

turn on optimization

Learn more about volatile keyword

- https://barrgroup.com/Embedded-Systems/How-To/C-Volatile-Keyword
- https://www.geeksforgeeks.org/understanding-volatile-qualifier-in-c/
- https://en.wikipedia.org/wiki/Volatile_(computer_programming)
-

Solution 2: tell compiler what registers are affected

inline8.c

```c
#include <stdio.h>

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

  scanf("%d %d", &a, &b);

  c = a+b;

  asm ("movl $1, %eax;"
          "movl $1, %ebx;"
          "movl $1, %ecx;"
          "movl $1, %edx");

  printf("a=%d b=%d, a+b=%d\n", a, b, c);

  return 0;
}
```

Extended Inline Assembly

K. N. Toosi
University of Technology

asm (“assembly code” : output registers : input registers : clobbered registers);

Solution 2: tell compiler what registers are affected

inline9.c

```
#include <stdio.h>

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

  scanf("%d %d", &a, &b);

  c = a+b;

  asm ("movl $1, %%eax;"
          "movl $1, %%ebx;"
          "movl $1, %%ecx;"
          "movl $1, %%edx;"  :    :    :   "eax", "ebx", "ecx", "edx");

  printf("a=%d b=%d, a+b=%d\n", a, b, c);

  return 0;
}
```

K. N. Toosi
University of Technology

Solution 2: tell compiler what registers are affected

inline9.c

```
#include <stdio.h>

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

  scanf("%d %d", &a, &b);

  c = a+b;

  asm ("movl $1, %%eax;"
          "movl $1, %%ebx;"
          "movl $1, %%ecx;"
          "movl $1, %%edx;"  :    :    :   "eax", "ebx", "ecx", "edx");

  printf("a=%d b=%d, a+b=%d\n", a, b, c);

  return 0;
}
```

use double %
for registers

Solution 2: tell compiler what registers are affected

K. N. Toosi
University of Technology

inline9.c

```c
#include <stdio.h>

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

  scanf("%d %d", &a, &b);

  c = a+b;

  asm ("movl $1, %%eax;"
       "movl $1, %%ebx;"
       "movl $1, %%ecx;"
       "movl $1, %%edx;"  :   :   :  "eax", "ebx", "ecx", "edx");

  printf("a=%d b=%d, a+b=%d\n", a, b, c);

  return 0;
}
```

clobbered registers

Solution 2: tell compiler what registers are affected

inline9.c

```c
#include <stdio.h>

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

  scanf("%d %d", &a, &b);

  c = a+b;

  asm ("movl $1, %%eax;"
       "movl $1, %%ebx;"
       "movl $1, %%ecx;"
       "movl $1, %%edx;"  :   :   :  "eax", "ebx", "ecx", "edx");

  printf("a=%d b=%d, a+b=%d\n", a, b, c);

  return 0;
}
```

```
b.nasihatkon@kntu:lecture13$ gcc -m32 inline9.c && ./a.out
2 3
a=2 b=3, a+b=5
b.nasihatkon@kntu:lecture13$ gcc -m32 -O1 inline9.c && ./a.out
2 3
a=2 b=3, a+b=5
```

turn on optimization

Solution 2: tell compiler what registers are affected

K. N. Toosi
University of Technology

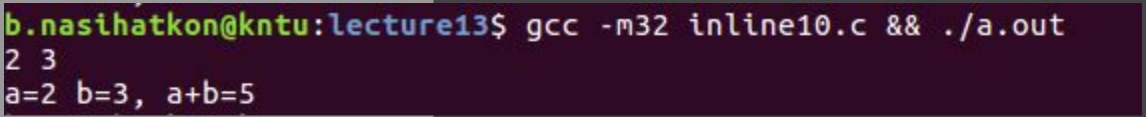
inline10.c

```
#include <stdio.h>

int main() {
  int a,b;

  register int c;

  scanf("%d %d", &a, &b);

  c = a+b;

  asm ("movl $1, %%eax;"
          "movl $1, %%ebx;"
          "movl $1, %%ecx;"
          "movl $1, %%edx;"  :    :    :   "eax", "ebx", "ecx", "edx");

  printf("a=%d b=%d, a+b=%d\n", a, b, c);

  return 0;
}
```

Solution 2: tell compiler what registers are affected

inline10.c

```c
#include <stdio.h>

int main() {
  int a,b;

  register int c;

  scanf("%d %d", &a, &b);

  c = a+b;

  asm ("movl $1, %%eax;"
         "movl $1, %%ebx;"
         "movl $1, %%ecx;"
         "movl $1, %%edx;"  :    :    :   "eax", "ebx", "ecx", "edx");

  printf("a=%d b=%d, a+b=%d\n", a, b, c);

  return 0;
}
```



```
b.nasihatkon@kntu:lecture13$ gcc -m32 inline10.c && ./a.out
2 3
a=2 b=3, a+b=5
```

Give input to inline assembly

inline11.c

```c
#include <stdio.h>
#include <string.h>

int main() {
  char msg[] = "Salaaaam Kako!\n";
  int  length = strlen(msg);

  asm ("movl   $4, %%eax;"  // system call 4: sys_write
       "movl   $1, %%ebx;"  // file handle 1: stdout
       "int    $0x80;"      // syscall
       :     : "c" (msg), "d" (length) : "eax", "ebx");

  return 0;
}
```

no outputs

inputs
(input constraints)

clobbered
registers

```
# sys_write
movl   $4, %eax   # syscall no.
movl   $1, %ebx   # file handle
movl   $msg, %ecx # message
movl   $13, %edx  # length
int    $0x80
```

Give input to inline assembly

inline11.c

```c
#include <stdio.h>
#include <string.h>

int main() {
  char msg[] = "Salaaaam Kako!\n";
  int  length = strlen(msg);

  asm ("movl   $4, %%eax;"  // system call 4: sys_write
       "movl   $1, %%ebx;"  // file handle 1: stdout
       "int    $0x80;"      // syscall
       :     : "c" (msg), "d" (length) : "eax", "ebx");

  return 0;
}
```

ecx

edx

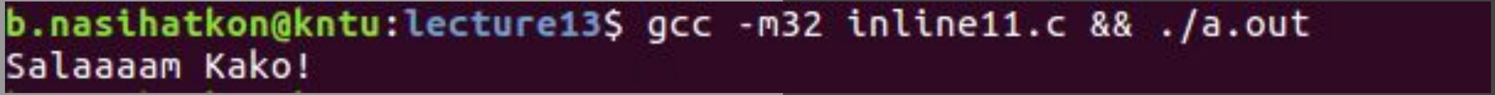
```
# sys_write
movl   $4, %eax   # syscall no.
movl   $1, %ebx   # file handle
movl   $msg, %ecx # message
movl   $13, %edx  # length
int    $0x80
```

Give input to inline assembly

inline11.c

```c
#include <stdio.h>
#include <string.h>

int main() {
  char msg[] = "Salaaaam Kako!\n";
  int  length = strlen(msg);

  asm ("movl   $4, %%eax;"  // system call 4: sys_write
       "movl   $1, %%ebx;"  // file handle 1: stdout
       "int    $0x80;"      // syscall
       :     : "c" (msg), "d" (length) : "eax", "ebx");




  return 0;
}
```

```asm
# sys_write
movl   $4, %eax   # syscall no.
movl   $1, %ebx   # file handle
movl   $msg, %ecx # message
movl   $13, %edx  # length
int    $0x80
```



```
b.nasihatkon@kntu:lecture13$ gcc -m32 inline11.c && ./a.out
Salaaaam Kako!
```

Registers

a	eax, ax, al
b	ebx, bx, bl
c	ecx, cx, cl
d	edx, dx, dl
S	esi, si
D	edi, di
r	register
f	a floating point register

Get output

inline12.c

```
#include <stdio.h>

int main() {
  int x = 12, y=13;

  printf("x=%d, y=%d\n", x,y);

  asm ("xchgl %%eax, %%ebx"
       : "=a" (x), "=b" (y)
       : "a"  (x), "b"  (y)
       :  );

  printf("x=%d, y=%d\n", x,y);

  return 0;
}
```

Get output

K. N. Toosi
University of Technology

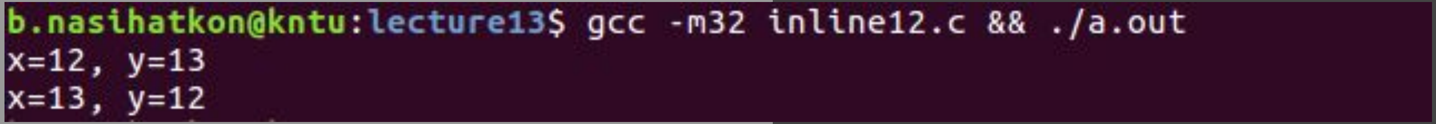
inline12.c

```c
#include <stdio.h>

int main() {
  int x = 12, y=13;

  printf("x=%d, y=%d\n", x,y);

  asm ("xchgl %%eax, %%ebx"
       : "=a" (x), "=b" (y) → outputs
       : "a"  (x), "b"  (y) → inputs
       :  );

  printf("x=%d, y=%d\n", x,y);

  return 0;
}
```

Get output

inline12.c

```
#include <stdio.h>

int main() {
  int x = 12, y=13;

  printf("x=%d, y=%d\n", x,y);

  asm ("xchgl %%eax, %%ebx"
        : "=a" (x), "=b" (y) → outputs
        : "a"  (x), "b"  (y) → inputs
        :  );

  printf("x=%d, y=%d\n", x,y);

  return 0;
}
```



```
b.nasihatkon@kntu:lecture13$ gcc -m32 inline12.c && ./a.out
x=12, y=13
x=13, y=12
```

Get output

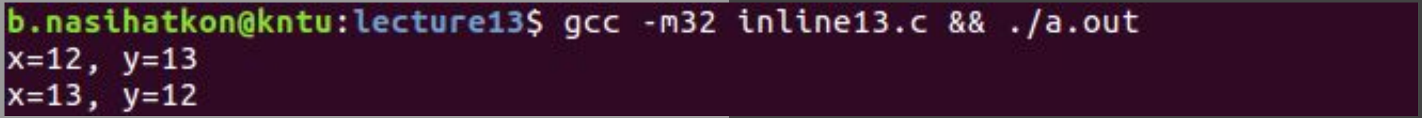
inline13.c

```
#include <stdio.h>

int main() {
  int x = 12, y=13;

  printf("x=%d, y=%d\n", x,y);

  asm ("xchgl %0, %1"
       : "=r" (x), "=r" (y) → outputs
       : "0"  (x), "1"  (y) → inputs
       :  );

  printf("x=%d, y=%d\n", x,y);

  return 0;
}
```

Get output

inline13.c

```
#include <stdio.h>

int main() {
  int x = 12, y=13;

  printf("x=%d, y=%d\n", x,y);

  asm ("xchgl %0, %1"
       : "=r" (x), "=r" (y) → outputs
       : "0"  (x), "1"  (y) → inputs
       :  );

  printf("x=%d, y=%d\n", x,y);

  return 0;
}
```



```
b.nasihatkon@kntu:lecture13$ gcc -m32 inline13.c && ./a.out
x=12, y=13
x=13, y=12
```

Use Intel Syntax with GCC

- Modern versions of GAS support Intel Syntax
- The GAS GNU Syntax is a bit different from NASM Syntax
  - the `.intel_syntax` and `.att_syntax` directives

Use Intel Syntax with GCC

- Put your code between the `.intel_syntax` (or `.intel_syntax noprefix`) and `.att_syntax` directives
- Better solution: Compile with -masm=intel gcc option.

Use Intel Syntax with GCC

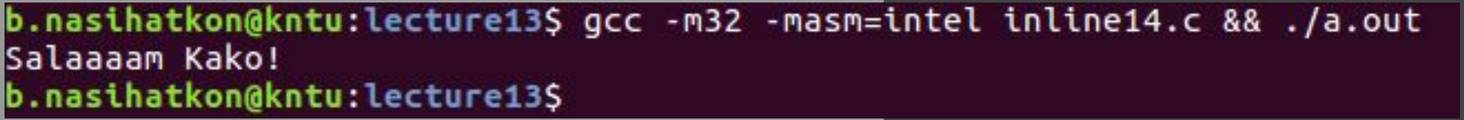
inline14.c

```c
#include <stdio.h>
#include <string.h>

int main() {
  char msg[] = "Salaaaam Kako!\n";
  int  length = strlen(msg);

  asm ("mov   eax, 4;"  // system call 4: sys_write
       "mov   ebx, 1;"  // file handle 1: stdout
       "int   0x80;"    // syscall
       :    : "c" (msg), "d" (length) : "eax", "ebx");

  return 0;
}
```

Use Intel Syntax with GCC

inline14.c

```
#include <stdio.h>
#include <string.h>

int main() {
  char msg[] = "Salaaaam Kako!\n";
  int  length = strlen(msg);

  asm ("mov   eax, 4;"  // system call 4: sys_write
       "mov   ebx, 1;"  // file handle 1: stdout
       "int   0x80;"    // syscall
       :     : "c" (msg), "d" (length) : "eax", "ebx");

  return 0;
}
```



```
b.nasihatkon@kntu:lecture13$ gcc -m32 -masm=intel inline14.c && ./a.out
Salaaaam Kako!
b.nasihatkon@kntu:lecture13$
```

Be careful with compiler optimization!

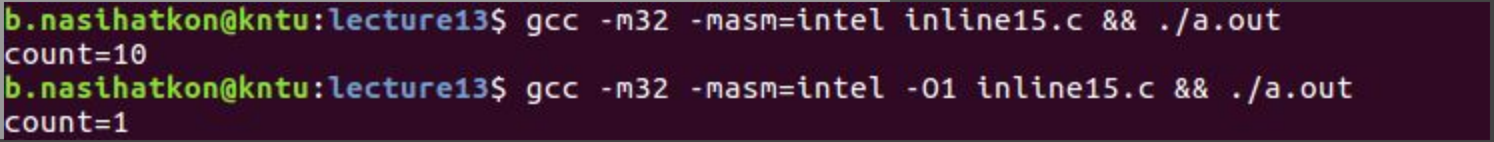
inline15.c

```c
#include <stdio.h>

int main() {

  int count = 0;

  asm ("mov eax, 0" : : : "eax");

  for (int i = 0; i < 10; i++) {
    asm ("inc eax;" : "=a" (count) : : );
  }

  printf("count=%d\n", count);

  return 0;
}
```

Be careful with compiler optimization!

inline15.c

```
#include <stdio.h>

int main() {

  int count = 0;

  asm ("mov eax, 0" : : : "eax");

  for (int i = 0; i < 10; i++) {
    asm ("inc eax;" : "=a" (count) : : );
  }

  printf("count=%d\n", count);

  return 0;
}
```



```
b.nasihatkon@kntu:lecture13$ gcc -m32 -masm=intel inline15.c && ./a.out
count=10
b.nasihatkon@kntu:lecture13$ gcc -m32 -masm=intel -O1 inline15.c && ./a.out
count=1
```

volatile keyword for inline assembly

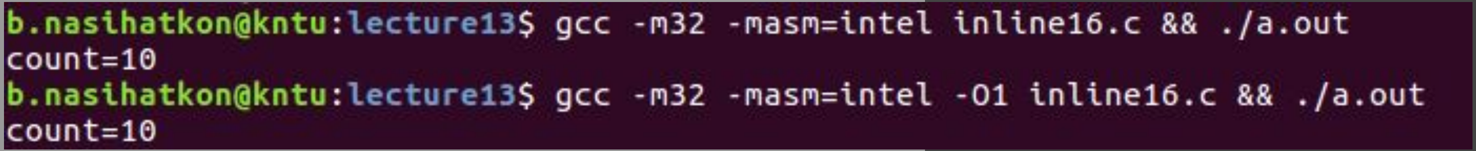
inline16.c

```
#include <stdio.h>

int main() {

  int count = 0;

  asm volatile ("mov eax, 0" : : : "eax");

  for (int i = 0; i < 10; i++) {
    asm  volatile ("inc eax;" : "=a" (count) : : );
  }

  printf("count=%d\n", count);

  return 0;
}
```

volatile keyword for inline assembly

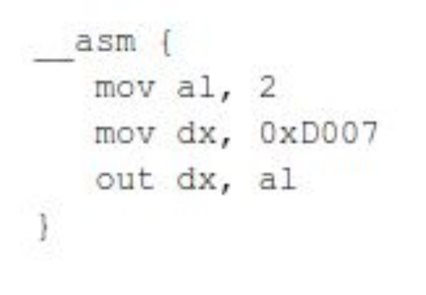
inline16.c

```c
#include <stdio.h>

int main() {

  int count = 0;

  asm volatile ("mov eax, 0" : : : "eax");

  for (int i = 0; i < 10; i++) {
    asm  volatile ("inc eax;" : "=a" (count) : : );
  }

  printf("count=%d\n", count);

  return 0;
}
```

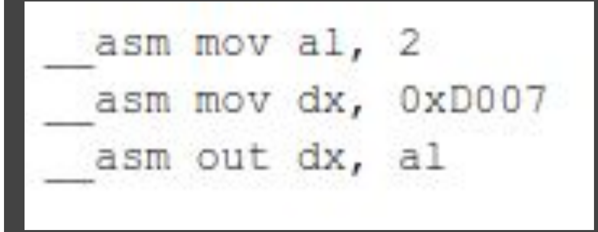


```
b.nasihatkon@kntu:lecture13$ gcc -m32 -masm=intel inline16.c && ./a.out
count=10
b.nasihatkon@kntu:lecture13$ gcc -m32 -masm=intel -O1 inline16.c && ./a.out
count=10
```

Inline assembly is compiler-dependent



```
__asm {
    mov al, 2
    mov dx, 0xD007
    out dx, al
}
```



```
__asm mov al, 2
__asm mov dx, 0xD007
__asm out dx, al
```

Microsoft Visual C

https://msdn.microsoft.com/en-us/library/45yd4tzz.aspx

References & further reading

- https://gcc.gnu.org/onlinedocs/gcc/Constraints.html
- https://www.codeproject.com/Articles/15971/Using-Inline-Assembly-in-C-C
- https://www.ibiblio.org/gferg/ldp/GCC-Inline-Assembly-HOWTO.html
- https://www.cs.virginia.edu/~clc5q/gcc-inline-asm.pdf