

GCC and operating systems

- a compiler and operating systems are often co-dependant
- during this weeks lecture we have examined the implementation of SYSTEM_TRANSFER in GNU LuK
- copy the following code into the C file suggested

GCC and operating systems

testgcc.c

```
void interruptsOn (void)
{
    asm volatile ("sti");
}

void interruptsoff (void)
{
    asm volatile ("cli");
}

unsigned int getFlags (void)
{
    unsigned int b;

    asm volatile("pushf ; popl %%ebx ; movl %%ebx, %[b]";
    return b;
}
```

GCC and operating systems



testgcc.c

```
typedef void *PROCESS;

int turnInterruptions (int oo)
{
    unsigned int flags = getFlags();
    if (oo)
        interruptsOn ();
    else
        interruptsoff ();

    return (flags & (1 << 9)) != 0;
}
```

GCC and operating systems

testgcc.c

```
void SYSTEM_TRANSFER (PROCESS *p1, PROCESS p2)
{
    int toOldState;

    toOldState = turnInterruptions (0);
    asm volatile ("pusha ; pushf"); /* push all register
    /* remember p1 is **void */
    asm volatile ("movl %[p1], %%eax ; movl %%esp, (%eax)
    :: [p1] "rm" (p1)); /* p1 := top of stack */
    asm volatile ("movl %[p2], %%eax ; movl %%eax, %%esp"
    :: [p2] "rm" (p2)); /* top of stack := p2 */
    asm volatile ("popf ; popa"); /* restore all registers */
    toOldState = turnInterruptions (toOldState);
}
```

GCC and operating systems

- compile this program using
 - ```
$ gcc -o testgcc-unoptimized.s -m32 -S testgcc.c
```
  - produce an instruction by instruction commentary of the function SYSTEM\_TRANSFER
    - the output of the compiler is placed into `testgcc-unoptimized.s`
- 
- now recompile the program using:
    - ```
$ gcc -O3 -o testgcc-optimized.s -m32 -S testgcc.c
```
 - the output from the compiler is placed into `testgcc-optimized.s`
 - produce an instruction by instruction commentary of the function SYSTEM_TRANSFER
 - which is easier to understand, the optimised or unoptimised assembly output?

GCC and operating systems