Основы работы с командной строкой Linux

```
sab@SAB:.../Lesson1$
```

nano Program.c

```c
#include <stdio.h>
int main()
{
        printf("Hello World!\n");
        return 0;
}
```

gcc Program.c –o Program

```
sab@SAB:.../Lesson1$ ./Program
Hello World!
```

Решим простую задачу

Написать процедуры для сложения и перемножения чисел
Вывести наименьший результат

Решим простую задачу

```c
#include <stdio.h>
#define X 10

void summ(int x, int y, int* sum) //comment
{
        *sum = x + y;
}

void mult(int x, int y, int* mult)
{
        *mult = x * y;
}

int main()
{
    int res_sum, res_mult;
    summ(10, X, &res_sum);
    mult(2, 2, &res_mult);
    if(res_sum < res_mult)
        printf("Sum = %d\n", res_sum);
    else
        printf("Mult = %d\n", res_mult);
    return 0;
}
```

Можно ли сделать лучше?

Разбиение программы на модули

lib.h

```
void summ(int, int, int*);
void mult(int, int, int*);
```

lib.c

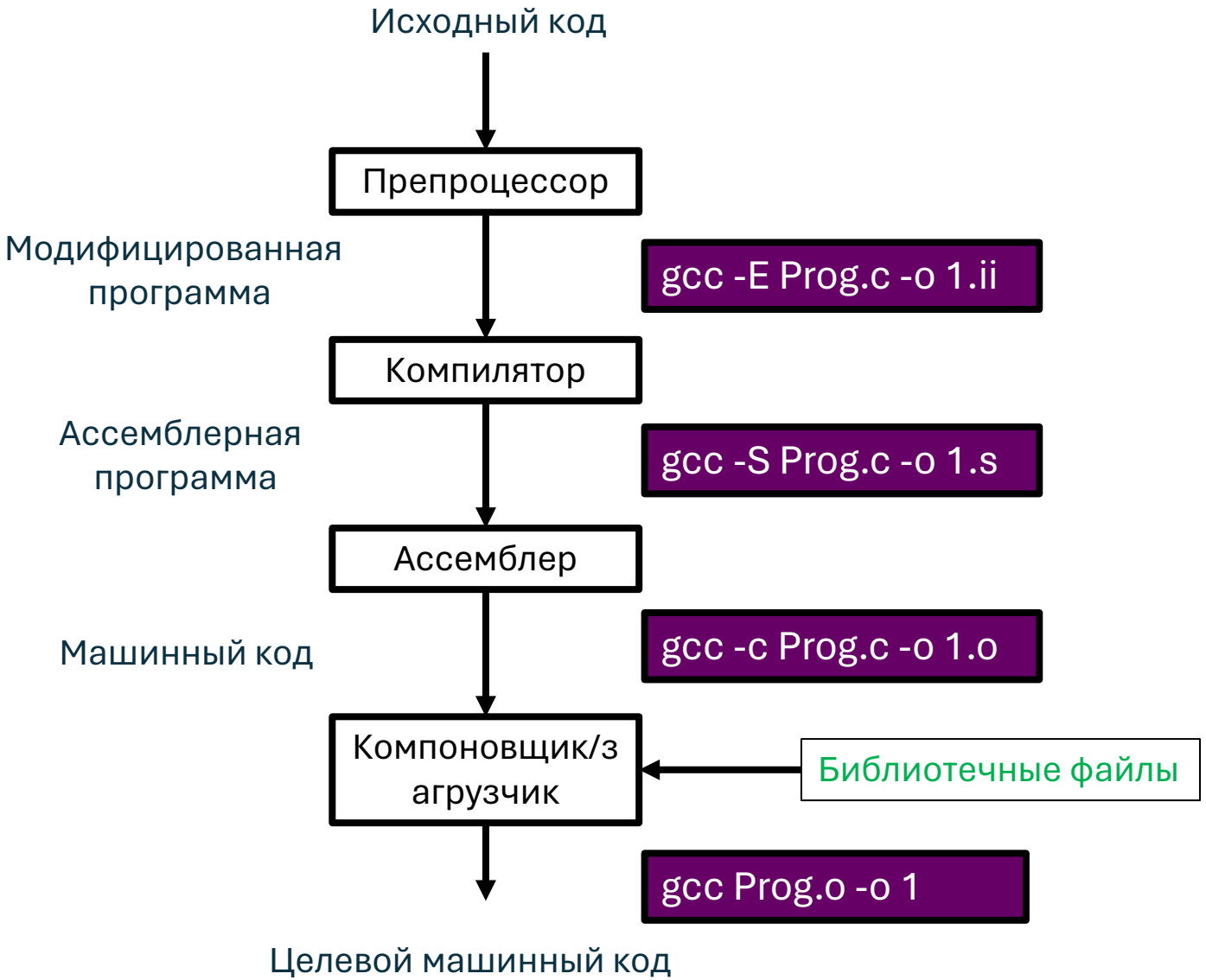
```
void summ(int x, int y, int* sum)
{
	*sum = x + y;
}

void mult(int x, int y, int* mult)
{
	*mult = x * y;
}
```

main.c

```
#include <stdio.h>
#include "lib.h"

int main()
{
int res_sum, res_mult;
summ(10, 10, &res_sum);
mult(2, 2, &res_mult);
if(res_sum < res_mult)
	printf("Sum = %d\n", res_sum);
else
	printf("Mult = %d\n", res_mult);
return 0;
}
```

Как правильно скомпилировать?

Этапы компиляции

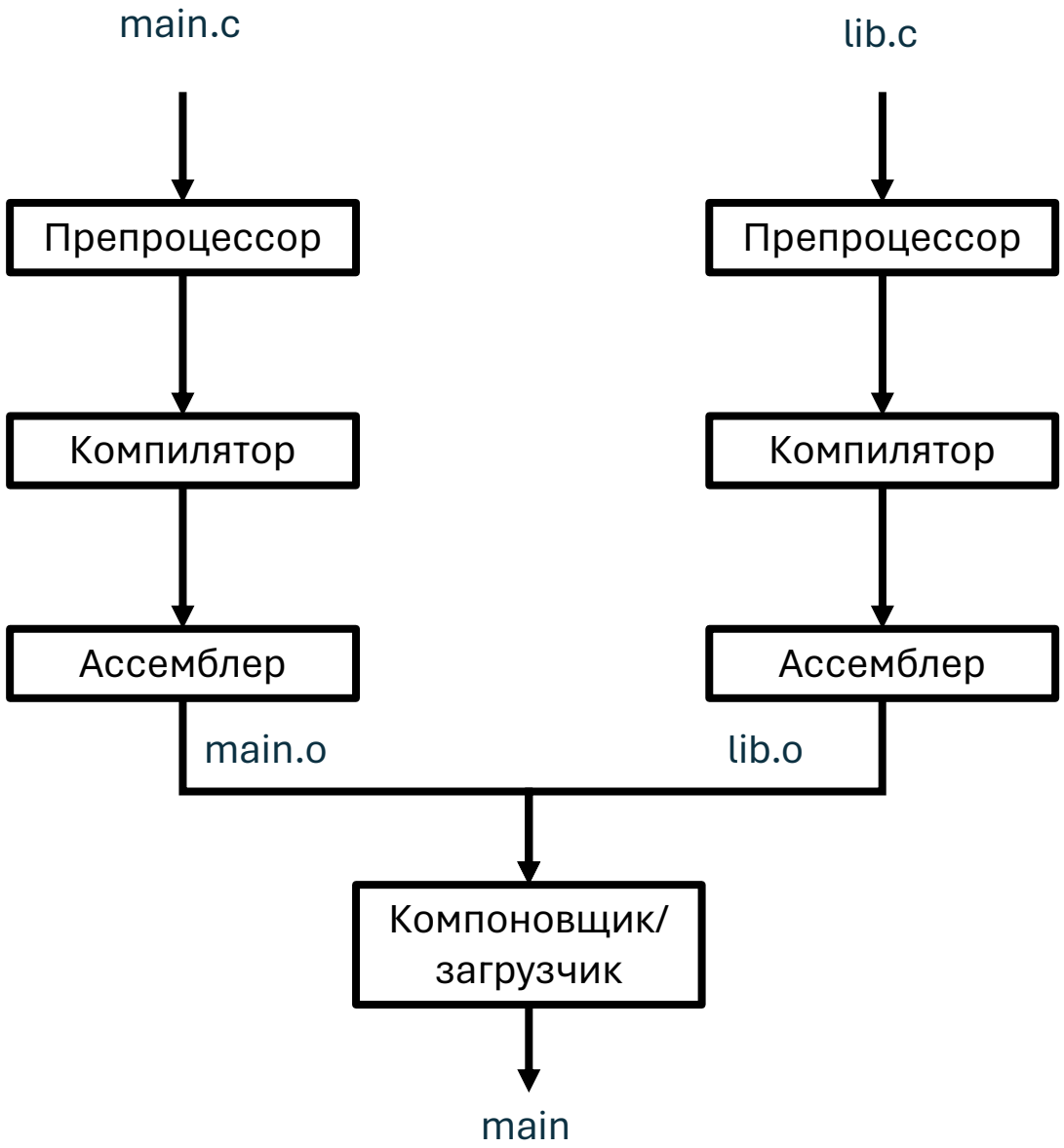


Кратко об ассемблере

Регистр	Назначение
%eax	хранение результатов промежуточных вычислений
%ebx	хранения адреса (указателя) на некоторый объект в памяти
%ecx	счетчик
%edx	хранения результатов промежуточных вычислений
%esp	содержит адрес вершины стека
%ebp	указатель базы кадра стека
%esi	индекс источника
%edi	индекс приёмника

Команда	Назначение
mov *источник, назначение*	копирование *источника* в *назначение*
lea *источник, назначение*	помещает адрес *источника* в *назначение*
add *источник*, приёмник	*приёмник* = *приёмник* + *источник*
sub *источник*, приёмник	*приёмник* = *приёмник* - *источник*
push *источник*	поместить в стек
pop *назначение*	извлечь из стека
cmp *операнд_2, операнд_1*	*операнд_1* – *операнд_2* и устанавливает флаги
jle *метка*	Переход если <=

Этапы компиляции



Разбиение программы на модули

lib.h

```
void summ(int, int, int*);
void mult(int, int, int*);
```

lib.c

```
void summ(int x, int y, int* sum)
{
	*sum = x + y;
}

void mult(int x, int y, int* mult)
{
	*mult = x * y;
}
```

main.c

```
#include <stdio.h>
#include "lib.h"

int main()
{
int res_sum, res_mult;
summ(10, 10, &res_sum);
mult(2, 2, &res_mult);
if(res_sum < res_mult)
	printf("Sum = %d\n", res_sum);
else
	printf("Mult = %d\n", res_mult);
return 0;
}
```

Как правильно скомпилировать?

```
gcc main.c -c -Werror -Wall -g -o main.o
gcc lib.c -c -Werror -Wall -g -o lib.o
gcc main.o lib.o -o main
./main
```

Разбиение программы на модули

lib.h

```c
void summ(int, int, int*);
void mult(int, int, int*);
```

lib.c

```c
void summ(int x, int y, int* sum)
{
	*sum = x + y;
}

void mult(int x, int y, int* mult)
{
	*mult = x * y;
}
```

main.c

```c
#include <stdio.h>
#include "lib.h“

int main()
{
int res_sum, res_mult;
summ(10, 10, &res_sum);
mult(2, 2, &res_mult);
if(res_sum < res_mult)
	printf("Sum = %d\n", res_sum);
else
	printf("Mult = %d\n", res_mult);
return 0;
}
```

```
gcc main.c -c -Werror -Wall -g -o main.o
gcc lib.c -c -Werror -Wall -g -o lib.o
gcc main.o lib.o -o main
./main
```

Что будет, если подключить заголовочный файл несколько раз?

Директивы препроцессора

Как избежать ошибки с многократным включением .h файла?

```
#define X
#ifdef X
/* Если до этого символ X был определён, то
включить текст до #endif. */
#endif

#ifndef X
/* Если до этого символ X НЕ был определён, то
включить текст до #endif. */
#endif
```

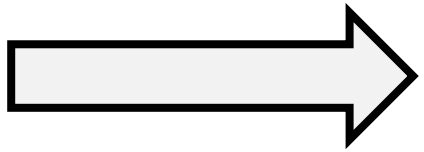
```c
#include <stdio.h>
#define X
int main()
{
    int a = 0;
#ifdef X
    a = 10;
#endif

#ifndef X
    a = 20;
#endif

    printf("%d\n", a);
    return 0;
}
```

```
gcc -E prog.c -o prog.ii
```

Директивы препроцессора

Как избежать ошибки с многократным включением .h файла?

```
void summ(int, int, int *);
void mult(int, int, int *);
```



```
#ifndef FILE_H
#define FILE_H
void summ(int, int, int *);
void mult(int, int, int *);
#endif
```

Make файлы

цель: зависимости

[tab] команда

```
gcc main.c -c -o main.o
gcc lib.c -c -o lib.o
gcc main.o lib.o -o main
```

makefile

```
all: clean
	gcc -c *.c
	gcc *.o -o main
clean:
	rm -f *.o
```

```
CFLAGS=-Wall -g -Werror
all: clean
	gcc -c *.c
	gcc *.o $(CFLAGS) -o main
clean:
	rm -f *.o
```

Разбиение программы на модули

lib.h

```
#pragma once
void summ(int, int, int*);
void mult(int, int, int*);
```

lib.c

```
void summ(int x, int y, int* sum)
{
	*sum = x + y;
}

void mult(int x, int y, int* mult)
{
	*mult = x * y;
}
```

main.c

```
#include <stdio.h>
#include "lib.h“

int main()
{
int res_sum, res_mult;
summ(10, 10, &res_sum);
mult(2, 2, &res_mult);
if(res_sum < res_mult)
	printf("Sum = %d\n", res_sum);
else
	printf("Mult = %d\n", res_mult);
return 0;
}
```

```
gcc main.c -c -Werror -Wall -g -o main.o
gcc lib.c -c -Werror -Wall -g -o lib.o
gcc main.o lib.o -o main
./main
```

Make файлы

цель: зависимости
[tab] команда

makefile

```
all: clean
	gcc -c *.c
	gcc *.o -o main
clean:
	rm -f *.o
```

```
gcc main.c -c -o main.o
gcc lib.c -c -o lib.o
gcc main.o lib.o -o main
```

```
CFLAGS=-Wall -g -Werror
all: clean
	gcc -c $(CFLAGS) *.c
	gcc *.o $(CFLAGS) -o main
clean:
	rm -f *.o
```

```
make
```

Задача на подумать

Модернизируйте Make скрипт из примера следующим образом:

1. Все файлы .c должны располагаться в директории src
2. Все объектные файлы должны быть перемещены в директорию o
3. Вынести название компилятора в качестве переменной