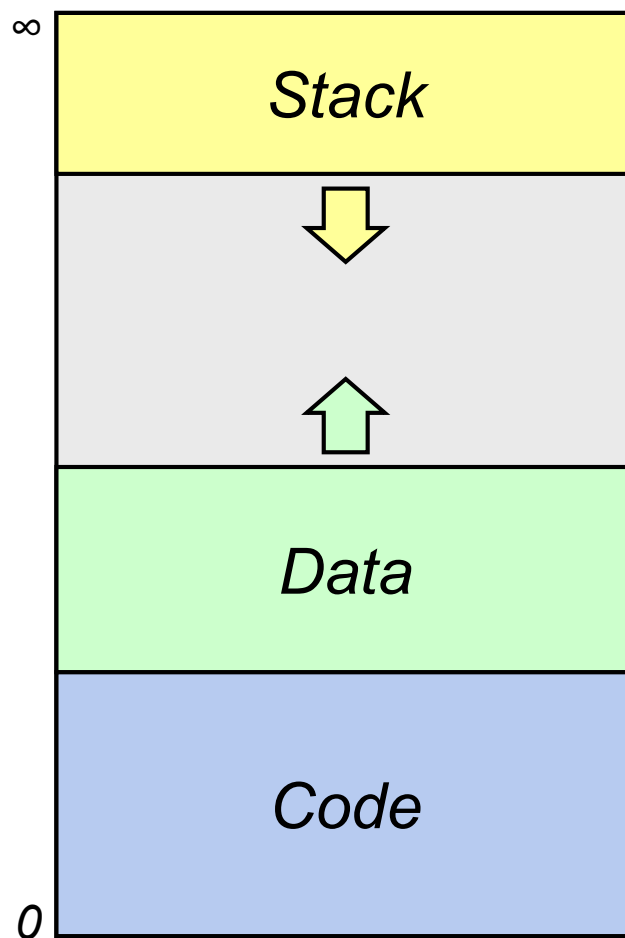


Създаване и стартиране на програми

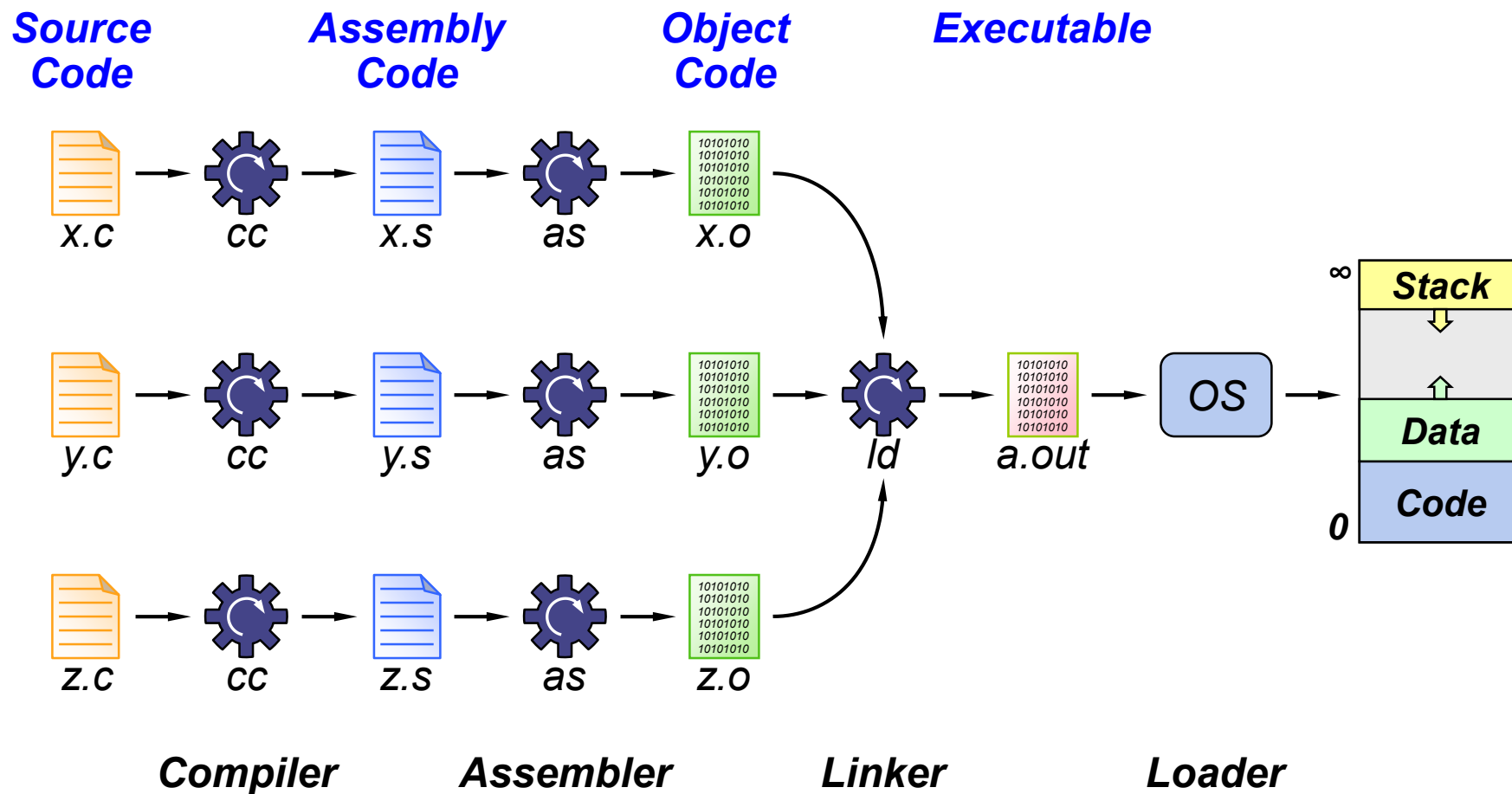
проф. д-р инж. Христо Вълчанов

<http://cs.tu-varna.bg>

Разпределение на паметта за процес



Създаване и изпълнение на програма



Компоненти

- **Асемблер (assembler)** – транслира възприемана от човека версия на машинните инструкции в машинен код, готов за директно изпълнение върху процесора.
- **Компилатор (compiler)** – транслира език от високо ниво в машинни инструкции.
- **Свързваща програма (linker)** – свързва модули с машинни инструкции в единен изпълним файл.

Формат на файл с обектен код

Необходимо е да включва:

- Машинен код на инструкциите
- Стойности на инициализирани данни
- Детайли на изисквано пространство на неинициализирани данни
- Дефинирани символи
- Недефинирани, но използвани символи
- Информация за преместването
- Входна точка

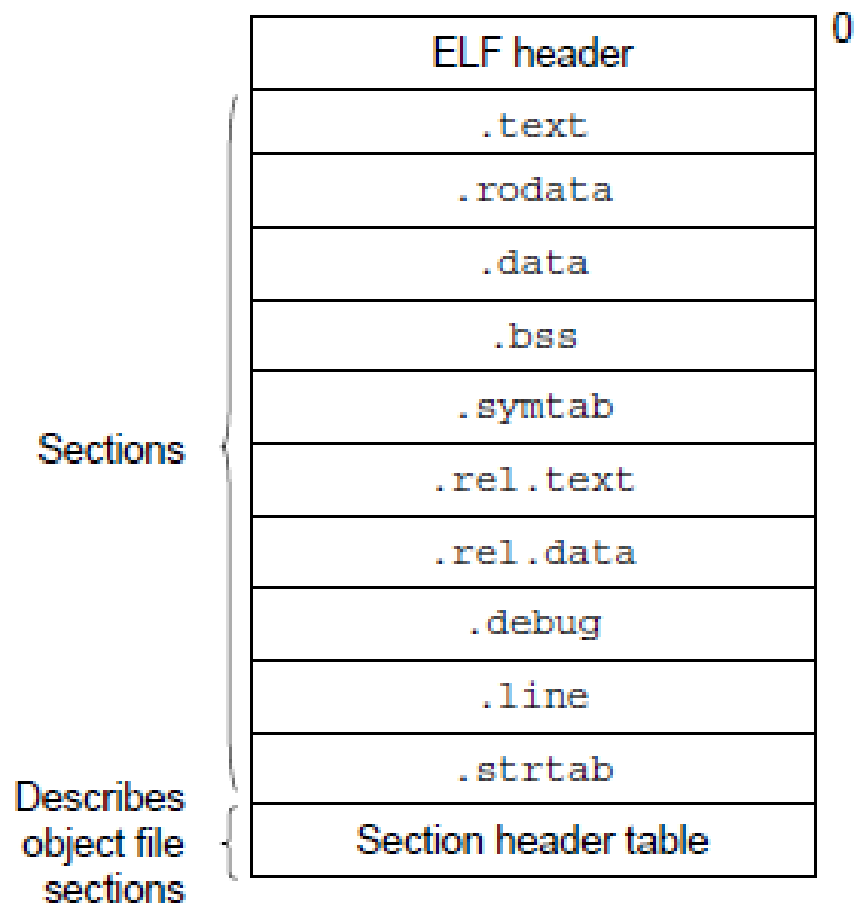
Видове обектни файлове

- Преместваем обектен файл (Relocatable object file) - .o
- Изпълним обектен файл (Executable object file) - a.out
- Споделен обектен файл (Shared object file) - .so

Формати обектен код

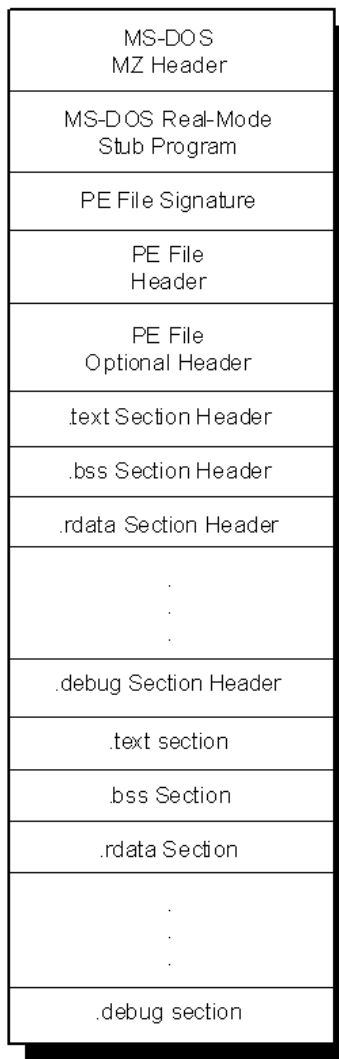
- Common Object File Format (COFF)
- Linux - Executable and Linking Format (ELF)
- Windows – Portable Executable (PE)

ELF формат



PE формат

PE File Format



Пример за разделна компиляция

main.c

```
int buf[2] = {1, 2};

int main()
{
    swap();
    return 0;
}
```

swap.c

```
extern int buf[];

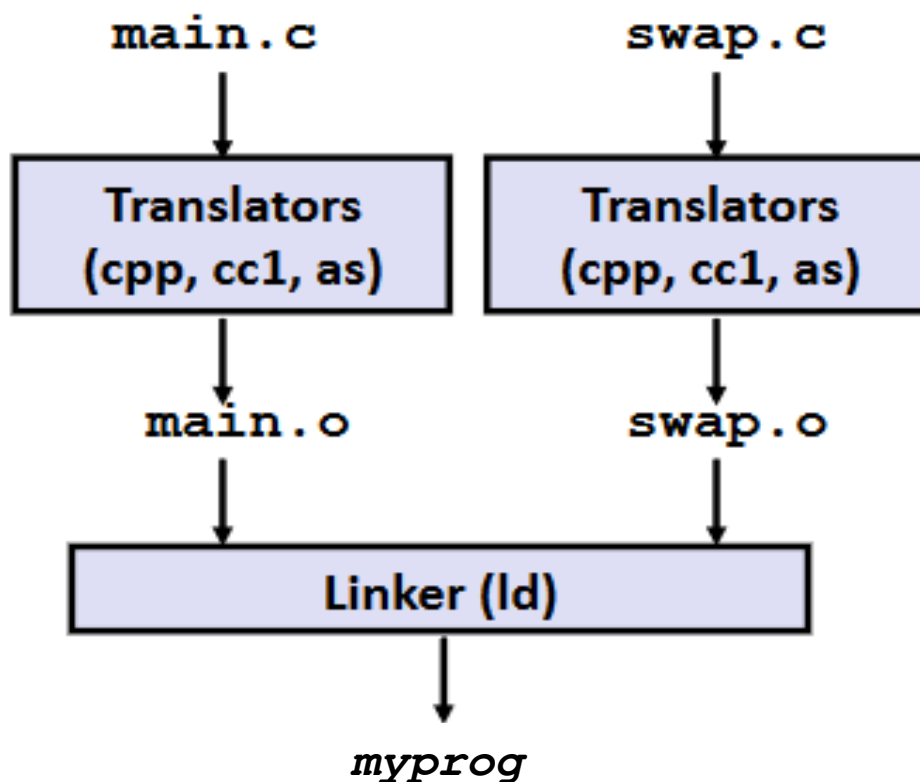
int *bufp0 = &buf[0];
static int *bufp1;

void swap()
{
    int temp;

    bufp1 = &buf[1];
    temp = *bufp0;
    *bufp0 = *bufp1;
    *bufp1 = temp;
}
```

Пример за разделна компилация

```
# gcc -o myprog main.c swap.c  
# ./myprog
```

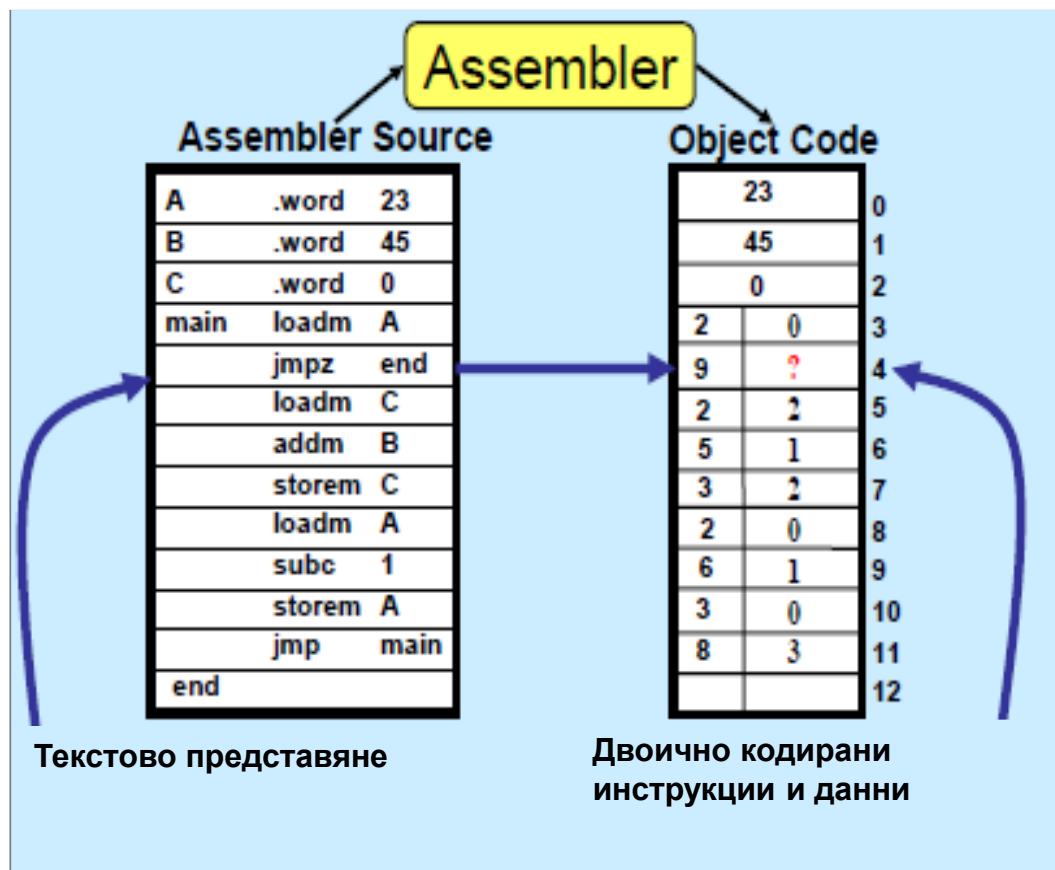


Асемблер

- **Асемблерен език** – текстово представяне на всяка машинна инструкция.
- Типично оперира в две фази:
 - Фаза 1 – изчислява необходимото пространство памет, изгражда таблица на символите.
 - Фаза 2 – обработва отново сорса, използвайки таблицата на символите като заменя символите със стойности.

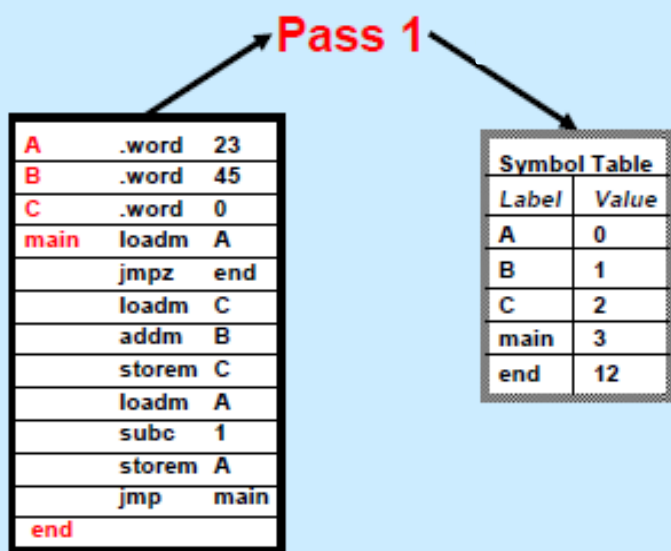
Функциониране на асемблер

```
A=23;  
B=45;  
C=0;  
int main() {  
    while (A>0) {  
        C=C+B;  
        A=A-1;  
    }  
}
```

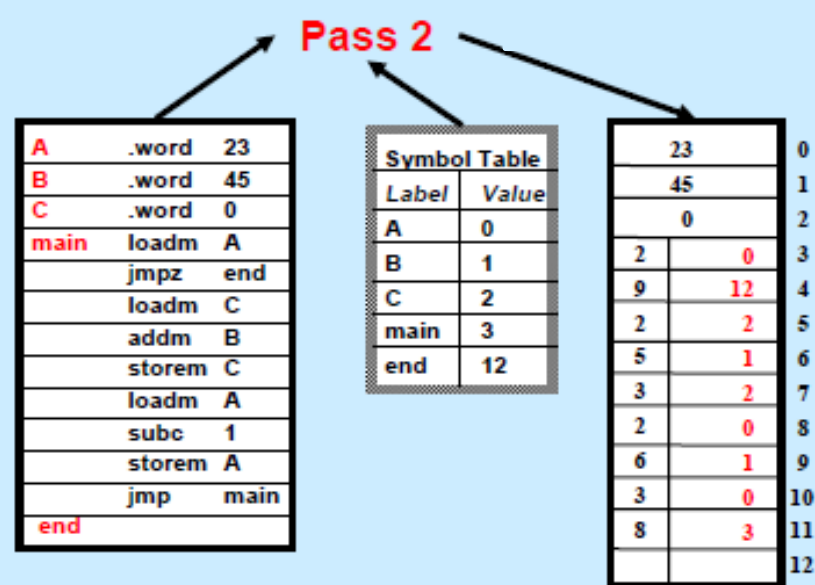


2- фазов асемблер

Изграждане на символна таблица



Генериране на обектния код

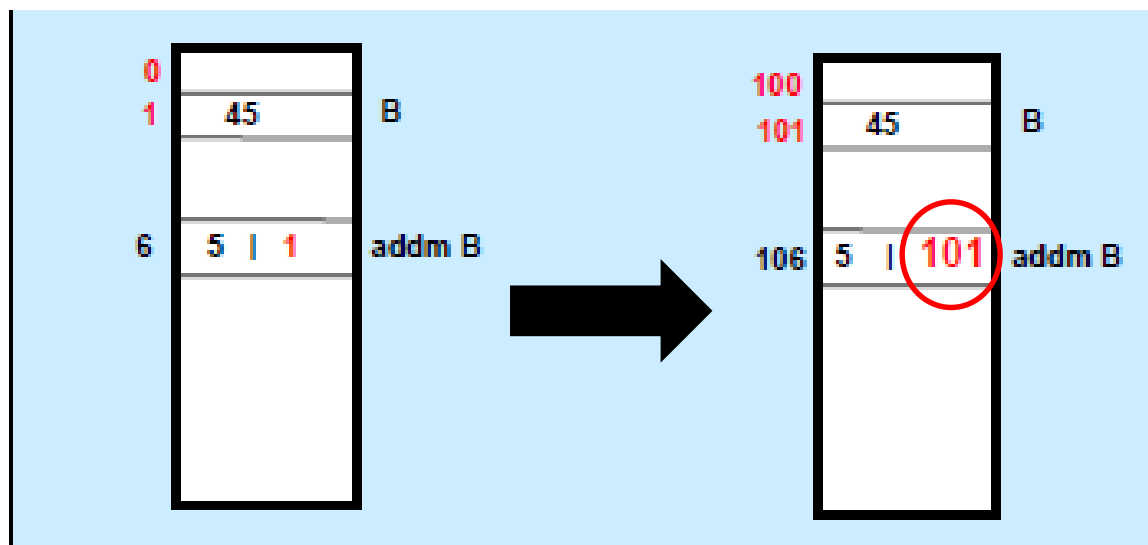


Свързващ редактор

- Свързва различни обектни модули в единен изпълним код, проверявайки за използването на символите.
- Реализира преместваемост на програмите – настройва референциите на символите съобразно нов начален адрес (***relocation***).

Преместване (relocation)

- Програмата е била асемблирана от начален адрес 0;
- Зарежда се в паметта от адрес 100 -> *relocation*.



Пренастройване на адреси

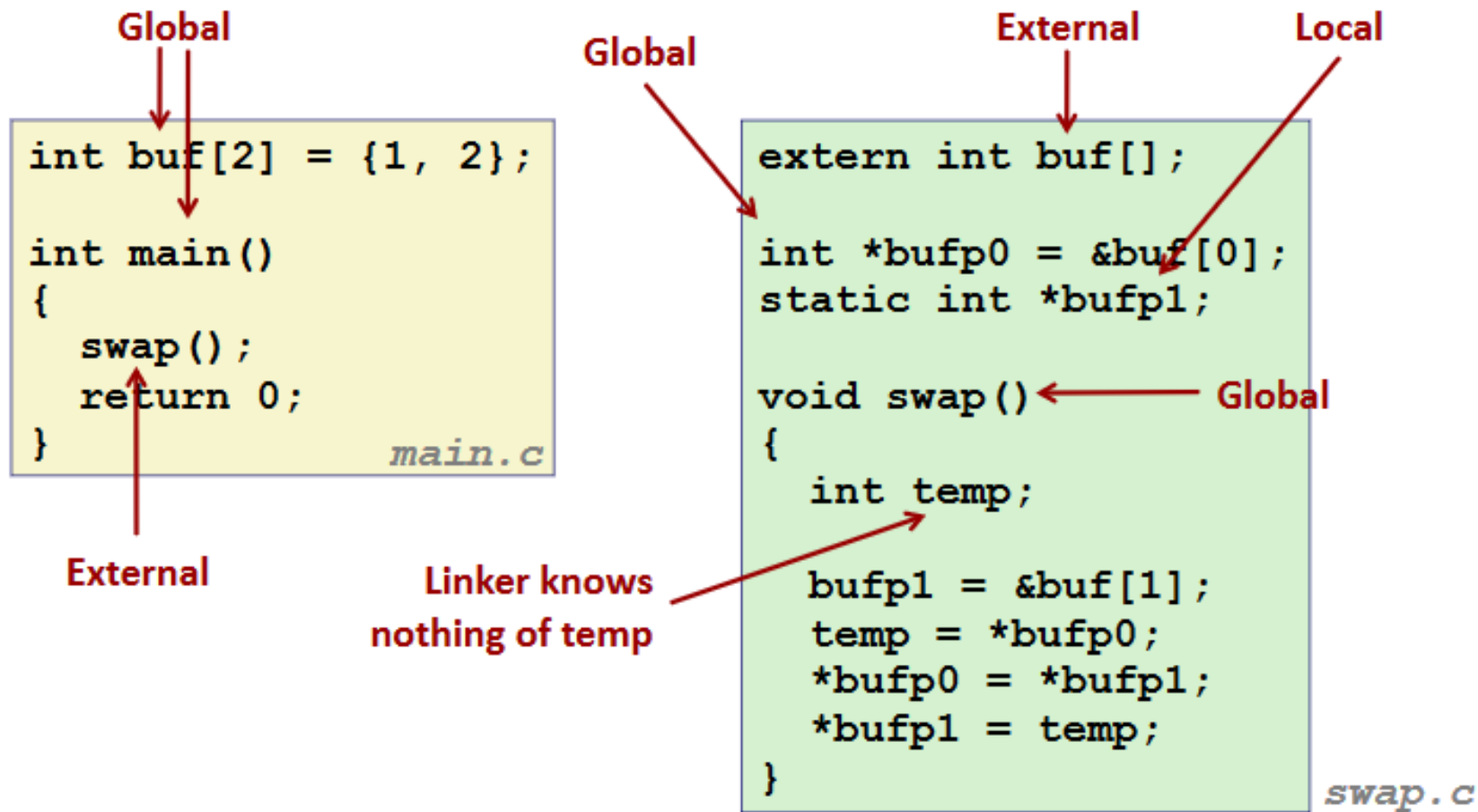
- Промяна на файла с обектния код

				Object Code File					
				Loc	Value		Relocate?		
A	.word	23		0	23		n	A	
B	.word	45		1	45		n	B	
C	.word	0		2	0		n	C	
main	loadm	A		3	2	0	y	main	
	jmpz	end		4	9	12	y		
	loadm	C		5	2	2	y		
	addm	B		6	5	1	y		
	storem	C		7	3	2	y		
	loadm	A		8	2	0	y		
	subc	1		9	6	1	n		
	storem	A		10	3	0	y		
	jmp	main		11	8	3	y		
end				12				end	

Видове символи

- **Глобални**, дефинирани в модул M, към които може да има обръщания от други модули;
- **Глобални**, към които има обръщания в модула M, но са дефинирани в други модули (*externals*);
- **Локални**, които са дефинирани и има обръщания към тях единствено в модула M.

Видове символи (2)



Разрешаване на симболи (Symbol Resolution)

```
void foo(void);
```



```
int main() {  
    foo();  
    return 0;  
}
```

```
unix> gcc -Wall -o linkerror linkerror.c  
/tmp/ccSz5uti.o: In function 'main':  
/tmp/ccSz5uti.o(.text+0x7): undefined reference to 'foo'  
collect2: ld returned 1 exit status
```

Множество дефинирани глобални СИМВОЛИ

Правило 1: Не се допускат множество *strong* символи.

Правило 2: Ако има *strong* и *weak* символ, избира се *strong*.

Правило 3: Ако има множество *weak* символи, избира се произволен.

Strong – Функции и инициализирани глобални променливи

Weak – Неинициализирани глобални символи

Множество дефинирани глобални СИМВОЛИ

```
/* foo.c */
```

```
int main() {  
    ...  
    return 0;  
}
```

```
/* bar.c */
```

```
int main() {  
    ...  
    return 0;  
}
```

```
unix> gcc foo.c bar.c
```

```
/tmp/cca015022.o: In function 'main':
```

```
/tmp/cca015022.o(.text+0x0): multiple definition of 'main'
```

```
/tmp/cca015021.o(.text+0x0): first defined here
```

Множество дефинирани глобални символи (2)

```
/* foo.c */  
int x = 1234;
```

```
int main() {  
    ...  
    return 0;  
}
```

```
/* bar.c */  
int x = 5678;
```

```
int test() {  
    ...  
    return 0;  
}
```

```
unix> gcc foo.c bar.c
```

```
/tmp/cca015022.o(.data+0x0): multiple definition of 'x'
```

```
/tmp/cca015021.o(.data+0x0): first defined here
```

Множество дефинирани глобални символи (3)

```
/* foo.c */  
void f(void);
```

```
int x = 1234;
```

```
int main() {  
    f();  
    printf("x=%d\n", x);  
    return 0;  
}
```

```
/* bar.c */  
int x;
```

```
void f() {  
    x = 5678;  
}
```

```
unix> gcc -o foobar foo.c bar.c  
unix> ./foobar  
x = 5678
```


Множество дефинирани глобални символи (4)

```
/* foo.c */  
void f(void);
```

```
int x;
```

```
int main() {  
    x = 1234;  
    f();  
    printf("x=%d\n", x);  
    return 0;  
}
```

```
/* bar.c */  
int x;
```

```
void f() {  
    x = 5678;  
}
```

```
unix> gcc -o foobar foo.c bar.c  
unix> ./foobar  
x = 5678
```

Свързване на модули

```
/* main.c */
```

```
int A=0;
```

```
int B=0;
```

```
int main() {
```

```
    A = incr(23);
```

```
    B = incr(45);
```

```
    return 0;
```

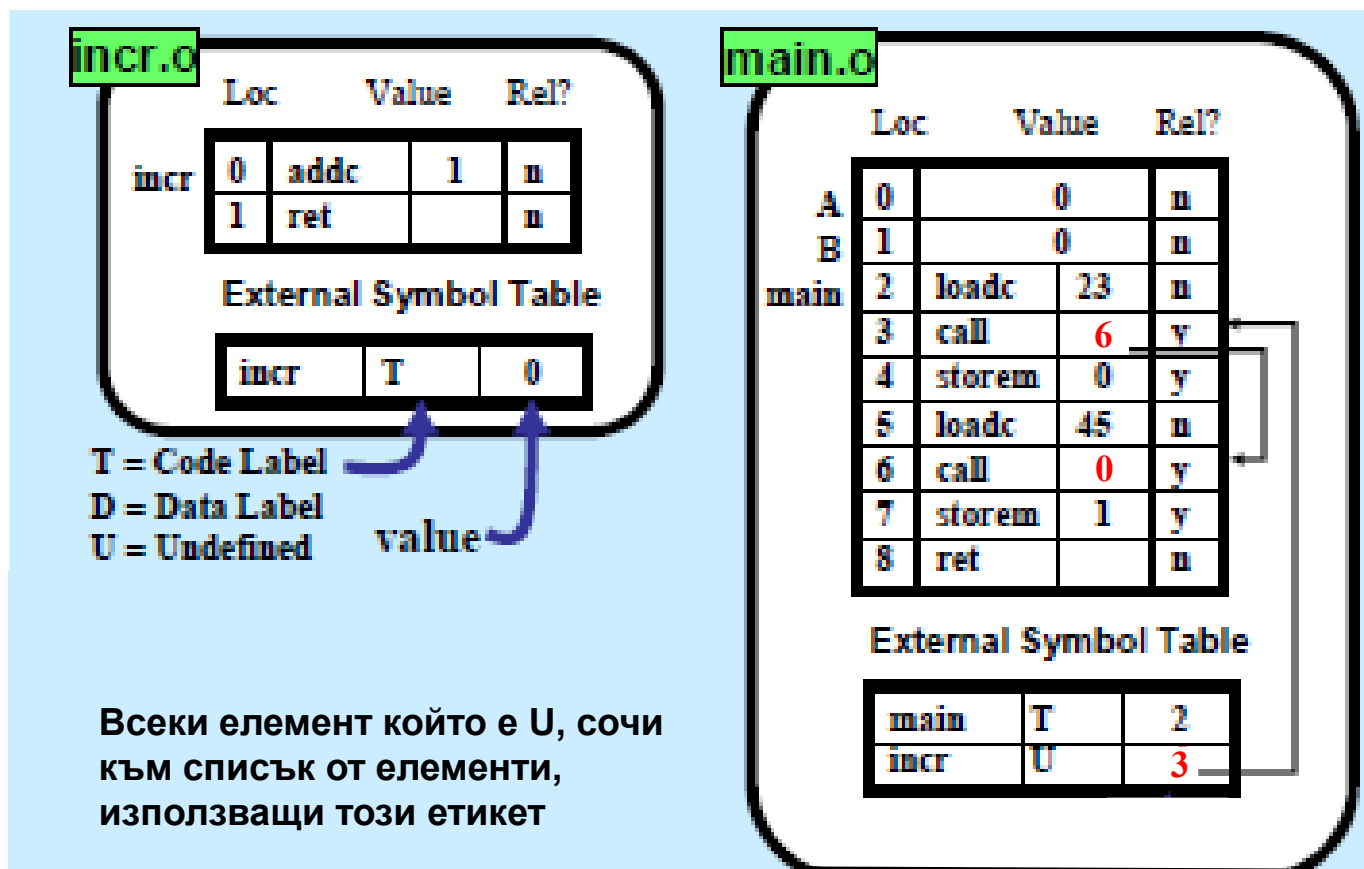
```
}
```

```
/* incr.c */
```

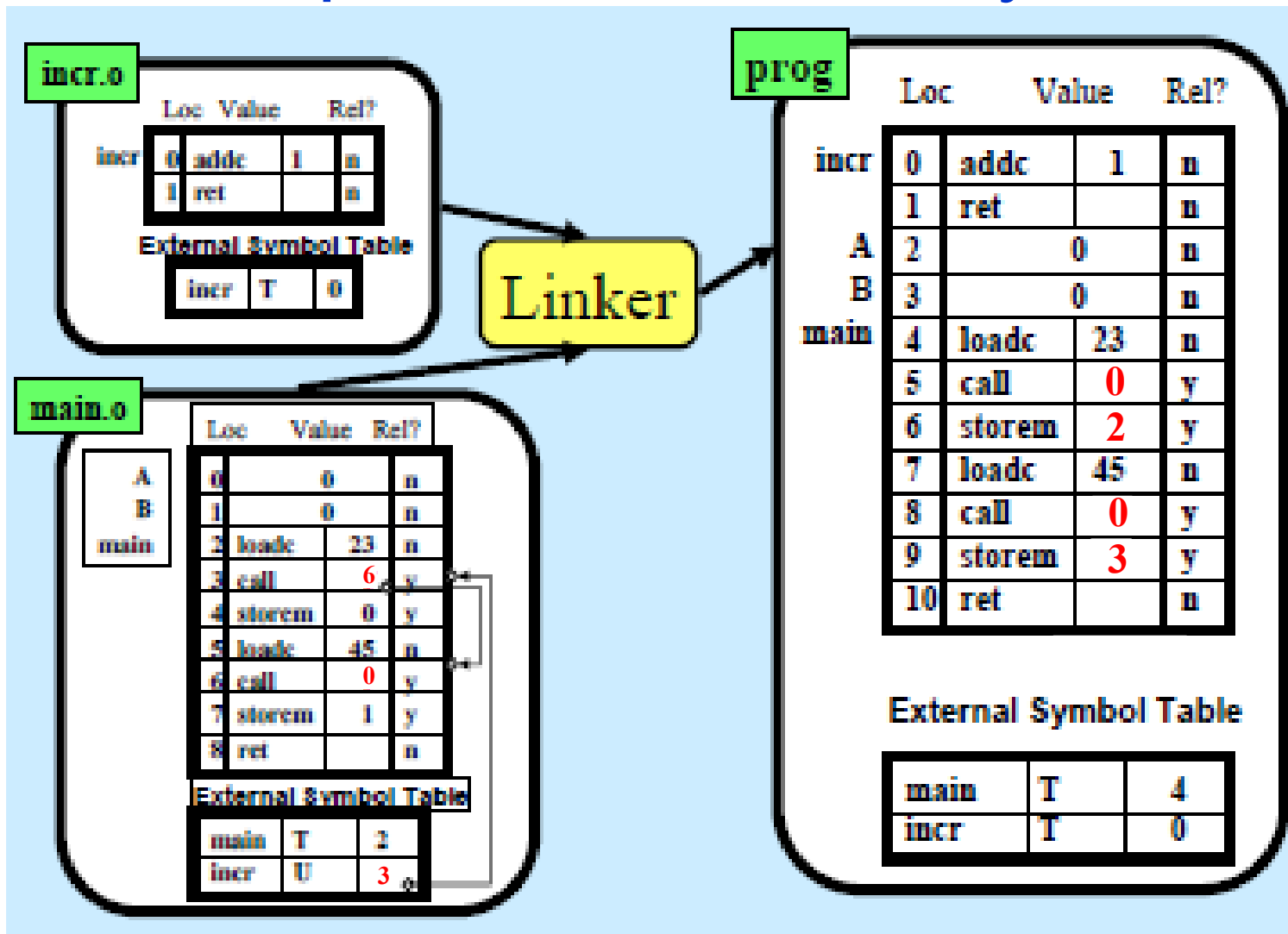
```
int incr(int x) {  
    return(x + 1);  
}
```

Свързване на модули

- Изграждане на таблица на външните символи



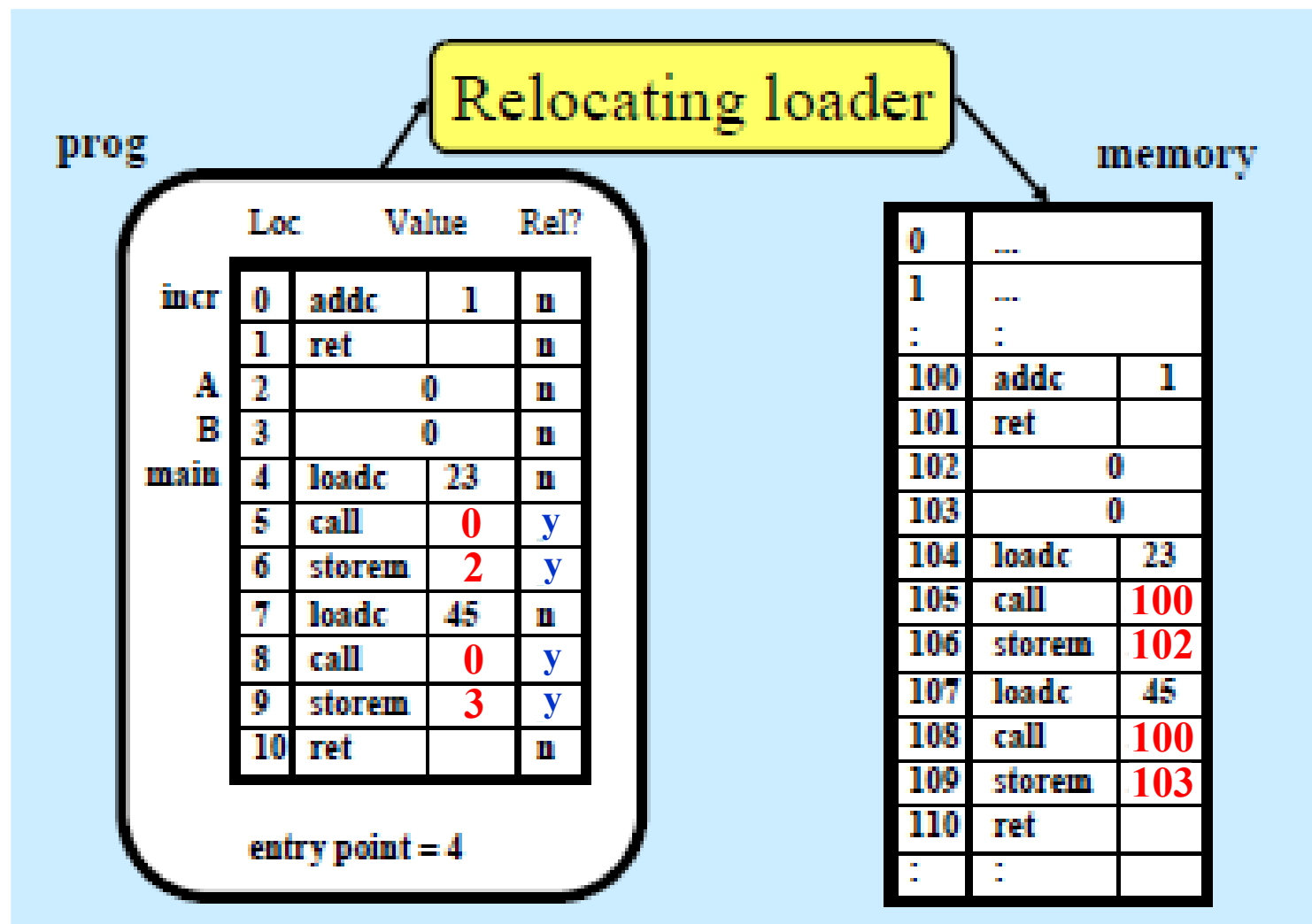
Свързан обектен модул



Зареждане на програма

- Ключова функция на ОС, реализирана от специален компонент – ***Loader***.
- Програмата се получава като обектен файл.
- Определяне на област от паметта където да се зареди (начален адрес).
- Пренастройване на адресите на вътрешните символи в обектния код.

Зареждане на програма



Въпроси?