Введение в архитектуру ЭВМ

Лекция 10. Раздельная трансляция и внешние символьные имена

План лекции

- Виды тестирования
  - Этапы цикла тестирования
- Схема организации связей между модулями
- Принцип работы редактора связей
- Пример раздельной трансляции
- Адресные значения внешних символьных имен, помещенные в команды ЯКЦП редактором связей
- Утилита `make`

Виды тестирования

1. **Автономное тестирование — одна функция.**

   `test.S + func.S`

2. **Тестирование подсистемы.**

   `test.S + func-1.S + func-2.S + … func-n.S`

3. **Комплексное тестирование.**

   `test-1.S + test-2.S + … test-n.S + ВСЕ func.S`

Средний продукт содержит до 10^4 модулей/функций.

Этапы цикла тестирования

A. **Трансляция исходных текстов всех модулей и их объединение (сборка).**

B. **Тестовый запуск.**

C. **Поиск и обнаружение одной, максимум двух ошибок.**

D. **Исправление исходных текстов модулей с ошибками.**

E. **Идти к A.**

NBNB. Такая трансляция всех исходных текстов может занимать несколько часов.

NBNB. В продукте могут быть сотни и тысячи ошибок (bugzilla)

NBNB. Таким образом, стоит технологическая проблема радикального ускорения сборки.

Решение этой проблемы - исключение необходимости трансляции исходных модулей в которых не исправлялись ошибки в пункте D цикла тестирования.

Проблема решается путём введения понятия «объектный модуль» (файлы с расширением „o“) — своеобразного «сборочного полуфабриката» и инструмента связывания объектных модулей в исполняемый файл — редактора связей.

Объектный файл содержит полностью оттранслированные команды на ЯКЦП и информацию о необходимых связях с другими объектными файлами, т. н. внешние символьные имена.

NBNB. Поскольку объектные файлы уже практически оттранслированы и при их связывании работу по трансляции выполнять не надо, редактор связей должен установить только связи между ними. Поэтому он работает на 3-4 порядка быстрее, чем ассемблер или другой компилятор.

Раздельной называется трансляция (в нашем случае ассемблирование) когда каждый исходный модуль находится в отдельном файле и транслируется в объектный файл отдельной командой (в нашем случае `as`). Сборка объектных модулей осуществляется редактором связей `ld`.

NBNB. Такое технологическое решение позволяет повторно оттранслировать только те исходные файлы где были исправлены ошибки и быстро собрать редактором связей обновленный исполняемый файл используя объектные файлы перетранслированных файлов и остальные объектные файлы, полученные при предыдущих трансляциях.

Т.е. в каждом цикле тестирования необходима трансляция только тех исходных файлов, где исправлены ошибки.

Отметим что система автоматизации сборки продуктов `make`, которую рассмотрим в конце лекции, существенно упрощает работу по сборке, автоматически определяя какие исходные файлы нужно перетранслировать.

Схема организации связей между модулями

В лекции 2, в разделе «Символьные имена», было отмечено, что символьные имена — метки и символьные имена с произвольными значениями, определенные в исходном тексте, получают числовые `адресные или абсолютные значения` при ассемблировании. Затем, при генерации ассемблером соответствующей двоичной команды ЯКЦП, эти значения помещаются в соответствующее поле этой команды (например в поле «Смещение»).

NBNB. При раздельной трансляции модулей может случиться так, что символьное имя определено в одном исходном файле, а используется в команде ассемблера в другом исходном файле, в котором оно не определено.

В этом случае значение символьного имени в том модуле, где оно не определено, НЕИЗВЕСТНО**, т. е. не может быть использовано при формировании соответствующей команду ЯКЦП. Это, в свою очередь, не позволяет корректно завершить ассемблирование.**

Эта проблема решается путём введения внешних и внутренних символьных имён.

Определение.

Внутренним **называется символьное имя, которое используется только в том исходном файле, в котором оно определено.**

Определение.

Внешним **называется символьное имя, которое определено в одном исходном файле, а используется хотя бы в одном другом исходном файле (при этом имя также называют** внешней ссылкой**).**

NBNB. В том исходном файле, где внешнее имя **определяется, оно должно быть описано как минимум дважды: один раз с помощью директивы** .global **и второй раз — для определения его характеристик и, возможно, значений, например, с помощью директив определения данных или задания меток.**

В том исходном файле, где внешнее имя только используется, оно должно быть описано только один — раз с помощью директивы .global**. Отметим, что для совместимости с другими ассемблерами**

допустимы написания `.globl` **и** `.global`.

NBNB. Внешние имена и их значения в отличие от внутренних имен, **доступны редактору связей** `ld`.

Принцип работы редактора связей

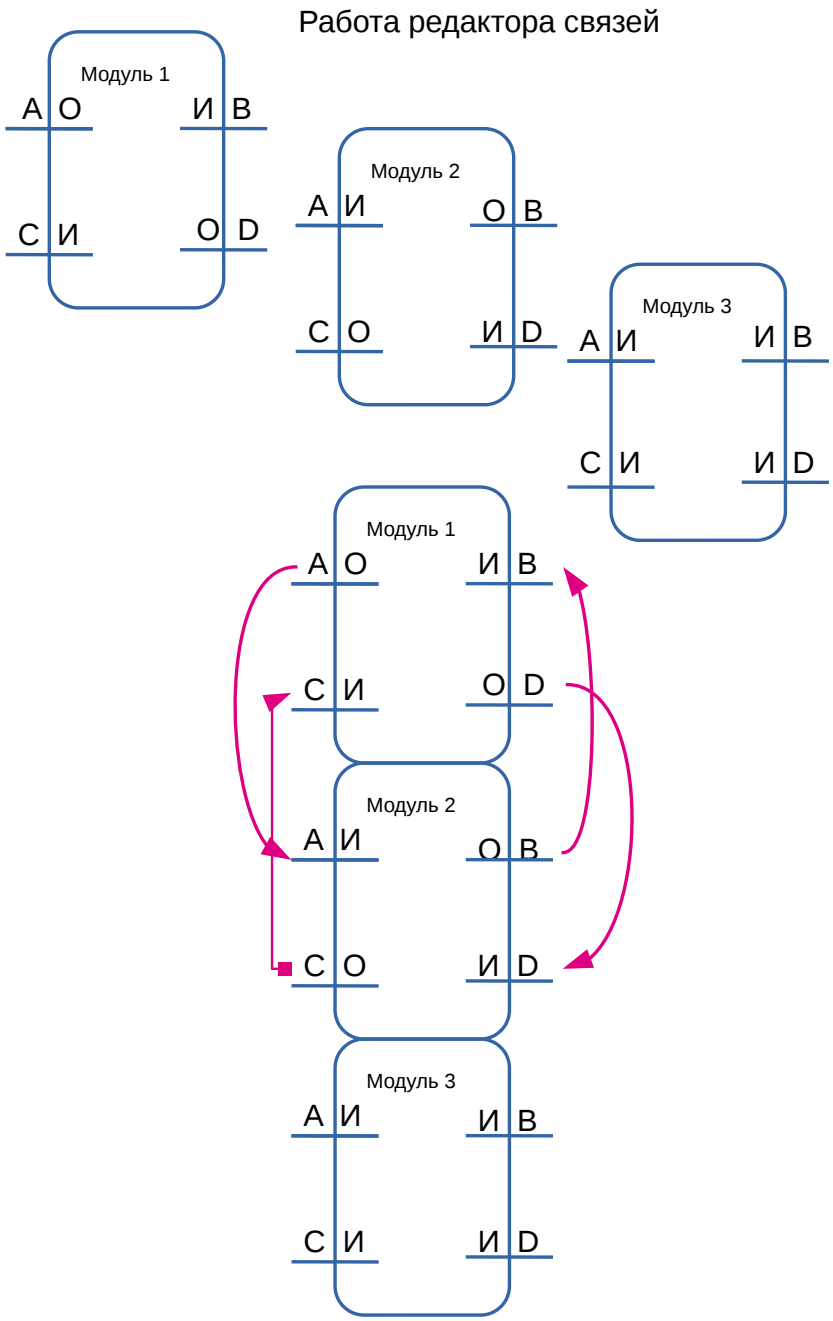

Рис. 1. Идея работа редактора связей.

Пока неизвестно относительное расположение модулей в исполняемом файле, нельзя вычислить числовое значение внешнего

символьного имени — которое есть расстояние в байтах от начала всех последовательно расположенных в памяти объектных файлов до адреса ОП в том объектном файле где внешнее имя определено. Однако если последовательно расположить объектные модули в ОП то это значение (по сути — перемещение) внешнего имени в исполняемом файле легко вычисляется, как это показано на следующем рисунке.

На рисунке символьные имена `A, B, C` и `D` обозначены буквой `O`, если они определены в объектном файле и буквой `И` — если они только используются в объектном файле.

Первое, что делает `ld` — объединяет в ОП объектные модули управляемым образом (есть режим по умолчанию) в единый файл. После этого, как видно из рисунка, необходимые числовые значения внешних имен легко вычисляются и могут использоваться при генерации команд ЯКЦП в их полях.

NB Говорят, что редактор связей `разрешает внешние ссылки`.

Пример раздельной трансляции

Рассмотрим листинги модулей примера раздельной трансляции. В этом примере используются те же три функции `main`, `Read_Sym` и `Trans_Sym`, что и в примере организации функций. Они решают ту же самую задачу, только сейчас они находятся не в одном исходном файле, а в трех отдельных — `main.S`, `Read_Sym.S` и `Trans_Sym.S`. Для связи файлов в них добавлены директивы `.global`, описывающие внешние символьные имена.

```
  1
  2                #        ABI соглашения о вызовах функций
  3
  4                #    Байты из массива Symbols читаются по одному.
  5                #    Если прочтен код цифры, то он преобразуется
  6                #    в 4-байтовое целое, иначе в значение -1.
  7                #    Результат записывается в элементы массива Numbers.
  8
  9                    .include "my-macro"    # подключение файла с
макроопределениями
  1                // Макроопределение завершения работы
  2
  3                .macro Finish
  4                         movl $0, %ebx # first argument: exit code
  5                         movl $1, %eax # sys_exit index
  6                         int $0x80 # kernel interrupt
  7                .endm
  8
 10
 11                .data # секция данных
 12
 13                Symbols:
 14 0000 39314132         .asciz   "91A23B456C789"   # массив символьных
кодов
 14      33423435
 14      36433738
 14      3900
 15                #                   цифр и "не цифр"
 16
 17                #    для показа исходного состяния  стека
 18                #    и мест РОН в нем после pusha в основной программ
 19
 20 000e 494E4954     Ini: .ascii "INIT" # стек
 21 0012 61784620     EAXF:    .ascii "axF " # %eax
 22 0016 64694620     EDIF:    .ascii "diF " # %edi
 23 001a 62702D34     EBPm4:   .ascii "bp-4" # для показа %ebp через %esi
 24
 25                .bss # секция общей памяти
 26
 27                .lcomm   Numbers, 40   # массив 4-х байтовых знач. цифр
 28
 29                # ---- добавлено для раздельной трансляции ---
 30
 31                # Описание внешних (заданных в других файлах) символьн
 32
 33                .globl Read_Sym, Numbers, EBPm4
 34
 35                # ---- конец добавления ----------------- ---
 36
 37                .global _start # точка входа - глобальная метка
 38
 39                .text # секция команд процессора
 40
 41                _start:
 42
 43 0000 90              nop
```

```
  44
  45                #    Индикаторы исходных состояний
  46
GAS LISTING main.S                page 2


  47                #    Стека
  48
  49 0001 A10E0000          movl Ini, %eax
  49      00
  50 0006 890424            movl %eax, 0(%esp)
  51
  52                #    Регистров общего назначения перед pusha
  53
  54 0009 A1120000          movl EAXF,  %eax  # первый
  54      00
  55 000e 8B3D1600          movl EDIF,  %edi  # последний
  55      0000
  56 0014 8B350000          movl EBPm4, %esi  # следующий после %ebp
  56      0000
  57                                  # !!! %ebp НЕ ТРОГАТЬ !!!
  58
  59 001a 60                pusha     # РОН в стек
  60
  61 001b 68000000          pushl $Symbols    # Параметр-2 - адрес массива в
стек
  61      00
  62 0020 6A08              pushl $8 # Параметр-1 в стек, цикл 0-7
  63
  64 0022 E8FCFFFF          call Read_Sym   # вызов функции
  64      FF
  65
  66 0027 83C408            addl $8,%esp  # очистить стек от пар. Read_Sym
  67
  68 002a 61                popa          # восстановить РОН
  69
  70                    Finish # конец работы, возврат в ОС
  70 002b BB000000      >  movl $0,%ebx
  70      00
  70 0030 B8010000      >  movl $1,%eax
  70      00
  70 0035 CD80          >  int $0x80
  71
  72                    .end   # последняя строка исходного текста

GAS LISTING main.S                page 3


DEFINED SYMBOLS
                main.S:13     .data:0000000000000000 Symbols
                main.S:20     .data:000000000000000e Ini
                main.S:21     .data:0000000000000012 EAXF
                main.S:22     .data:0000000000000016 EDIF
                main.S:23     .data:000000000000001a EBPm4
                main.S:27     .bss:0000000000000000 Numbers
                main.S:41     .text:0000000000000000 _start
```

```
UNDEFINED SYMBOLS
Read_Sym
```

Конец main.lst

```
GAS LISTING Read_Sym.S                  page 1


   1
   2                .# ---- добавлено для раздельной трансляции ---
   3
   4                .# Описание внешних  символьных имен
   5
   6                .globl Read_Sym, Trans_Sym, Numbers, EBPm4
   7
   8                .# ---- конец добавления ------------------ ---
   9
  10                .text    # секция команд процесора
  11
  12                .type    Read_Sym, @function    # читает Symbols в цикле
  13
  14                #    Имеет два параметра
  15
  16                # P1 - число байтов для чтение из массива
  17                # P2 - адрес массива откуда читать
  18
  19                # Прочтенный байт передается в Trans_Sym.
  20                # Ее результат возвращается в %eax и передается
  21                # в элементы массиве Numbers
  22
  23                Read_Sym:
  24
  25                #    Стандартный пролог
  26
  27 0000 55                pushl %ebp     # %ebp  вызывающей -> стек
  28 0001 89E5              movl  %esp, %ebp # обеспечить адресный доступ к
                    #                   параметрам и
  29                #                 локальным переменным в стеке путем
  30                #                 базовой адресации через ebp
  31
  32                .data # секция данных
  33
  34 0000 4C467231      LVAR1:   .ascii "LFr1" # показ локал. перем. Кадра 1
  35
  36                .text # секция команд процессора
  37
  38 0003 83EC04            subl  $4, %esp    # завести локал. перем. в
Кадре 1
  39 0006 A1000000            movl  LVAR1,%eax
  39      00
  40 000b 8945FC                movl  %eax,-4(%ebp)
  41
  42                #    Собственно Код  функции
  43
  44 000e 29C9              subl %ecx, %ecx   # иниц. цикла по байтам
Symbols
  45
```

```
 46                 #    Начало цикла
 47
 48                 NextSym:
 49 0010 8B550C             movl 12(%ebp), %edx  # адрес P2 - массива в %edx
 50
 51                 #    Подготовка вызова функции Trans_Sym
 52
 53                 #    Ее параметр - байт передадим через %bl рег. %ebx
 54
 55 0013 29DB                subl %ebx, %ebx   # все нули
 56
GAS LISTING Read_Sym.S                  page 2


 57                 #    - передадим код символа в %bl
 58                 #    %edx - базовый - взяли из P2,
 59                 #    %ecx - индексный - номер цикла, MM = 1 - один байт
 60                 #    регистровая адресация
 61
 62 0015 8A1C0A             movb (%edx,%ecx,1), %bl
 63
 64                 #    Параметр Trans_Sym готов, можно ее вызывать.
 65
 66                 .data # секция данных
 67
 68                 #    для показа мест РОН в стеке после pusha в ReadSym
 69
 70 0004 61786631    EAXf1:   .ascii "axf1" # %eax
 71 0008 64696631    EDIf1:   .ascii "dif1" # %edi
 72
 73                 .text # секция команд процессора
 74
 75                 #    Индикаторы РОН f1 перед pusha
 76
 77 0018 A1040000           movl EAXf1,  %eax # первый
 77      00
 78 001d 8B3D0800           movl EDIf1,  %edi # последний
 78      0000
 79 0023 8B350000           movl EBPm4,  %esi # следующий после %ebp
 79      0000
 80                                   # !!! %ebp НЕ ТРОГАТЬ !!!
 81
 82 0029 60                 pusha         # сохранить РОН функции Read_Sym
 83
 84 002a 53                 pushl %ebx    # Параметр Trans_Sym в стек
 85
 86 002b E8FCFFFF           call Trans_Sym
 86      FF
 87
 88 0030 83C404             addl $4,%esp  # очистить стек от параметров
                                     # Trans_Sym
 89
 90                 #    Опять работает Read_Sym
 91
 92                 #    В %eax 4 байтовый результат Trans_Sym
 93
 94                 #    Запись результата в массив Numbers.
```

```
 95
 96             #    Баз.  регистр НЕ ЗАДАН - запятая после лев. скобки
 97             #    %ecx - индексный регистр, масштаьный множитель - 4
 98             #    т.к. элементы Numbers - 4-х байтовые слова
 99             #    регистровая адресация
100
101 0033 89048D00         movl %eax, Numbers(,%ecx,4)
101      000000
102
103 003a 61              popa   # восстановть регистры Read_Sym
104
105 003b 41              incl %ecx     # наращиваем счетчик цикла
106 003c 3B4D08          cmpl 8(%ebp), %ecx   # счетчик цикла равен P1 ?
107
108 003f 75CF            jne NextSym        # ДА, на повтор



GAS LISTING Read_Sym.S                page 3


109                              # НЕТ - выходим из цикла
110
111             #    Стандартный эпилог функции
112
113 0041 89EC            movl %ebp, %esp # восстановить указатель стека
114 0043 5D              popl %ebp       # восстановить ebp
115 0044 C3              ret      # возврат в вызывающую
116
117                 .end   # последняя строка исходного текса

GAS LISTING Read_Sym.S                page 4


DEFINED SYMBOLS
          Read_Sym.S:23     .text:0000000000000000 Read_Sym
          Read_Sym.S:34     .data:0000000000000000 LVAR1
          Read_Sym.S:48     .text:0000000000000010 NextSym
          Read_Sym.S:70     .data:0000000000000004 EAXf1
          Read_Sym.S:71     .data:0000000000000008 EDIf1

UNDEFINED SYMBOLS
Trans_Sym
Numbers
EBPm4
```

Конец Read_Sym.lst

```
GAS LISTING Trans_Sym.S               page 1


   1
   2             #    Функция преобразования кода символа в числовое зна
   3             #    с фильтрацией кодов цифр. P1 - байт кода.
Возвращается
   4             #    значение цифры или -1 если код не символа цифры.
```

```
   5
   6                # ---- добавлено для раздельной трансляции ---
   7
   8                # Описание внешних  символьных имен
   9
  10                .globl Trans_Sym
  11
  12                # ---- конец добавления ----------------------
  13
  14                .type    Trans_Sym, @function
  15
  16                Trans_Sym:
  17
  18                #    Стандартный пролог фунции
  19
  20 0000 55                pushl %ebp     # сохранить в стеке значение,
бывшее                                    # в вызывающей
  21 0001 89E5              movl  %esp, %ebp # обеспечить адресный доступ к
                                          # параметрам и
  22                #                     локальным переменным в стеке путем
  23                #                     базовой адресации через %ebp
  24
  25                .data # секция данных
  26
  27                LVAR2:
  28 0000 4C467232          .ascii "LFr2" # показ локальной переменной Кадра
2
  29
  30                .text # секция команд процессора
  31
  32 0003 83EC04            subl  $4, %esp # завести локальн. перемен. Кадра
1
  33 0006 A1000000            movl  LVAR2,%eax
  33      00
  34 000b 8945FC              movl  %eax,-4(%ebp)
  35
  36                #    тело функции
  37
  38 000e 8B4508            movl 8(%ebp), %eax  # первый  параметр в eax
  39
  40                #    Фильтр кода символа цифры
  41
  42 0011 3C39              cmpb $'9', %al      # код больше кода символа '9'
?
  43 0013 7709              ja Ret_error      # ДА - на возврат -1
  44 0015 3C30              cmpb $'0', %al      # код меньше кода сивола
'0' ?
  45 0017 7205              jb Ret_error      # ДА - на возврат -1
  46
  47 0019 83E830            subl $0x30, %eax # получение числового значения
  48
  49 001c EB05              jmp Ret_norm   # на возврат числ. знач. цифры
  50
  51                Ret_error:
  52
  53 001e B8FFFFFF          movl $-1,%eax  # для возврата - код не цифры
  53      FF
```

```
  54
  55              Ret_norm:

GAS LISTING Trans_Sym.S                 page 2


  56
  57              #    Стандартный эпилог функции
  58
  59 0023 89EC             movl %ebp, %esp # восстановить указатель стека
  60 0025 5D               popl %ebp       # восстановить ebp
  61 0026 C3               ret      # возврат в вызывающую
  62
  63              #    Конец  Trans_Sym
  64
  65                   .end   # последняя строка исходного текста
GAS LISTING Trans_Sym.S                 page 3


DEFINED SYMBOLS
         Trans_Sym.S:16     .text:0000000000000000 Trans_Sym
         Trans_Sym.S:27     .data:0000000000000000 LVAR2
         Trans_Sym.S:51     .text:000000000000001e Ret_error
         Trans_Sym.S:55     .text:0000000000000023 Ret_norm

NO UNDEFINED SYMBOLS
```

Можно посмотреть как `ld` разрешил внешние ссылки (т. е. определил значения внешних символьных имен) выполнив для исполняемого файла команду `nm -n main`, которая выдает:

```
08048074 T _start
080480ab T Trans_Sym
080480c9 t Ret_error
080480ce t Ret_norm
080480d2 T Read_Sym
080480e2 t NextSym
08049117 d Symbols
08049125 d Ini
08049129 d EAXF
0804912d d EDIF
08049131 D EBPm4
08049135 d LVAR2
08049139 d LVAR1
0804913d d EAXf1
08049141 d EDIf1
08049145 B __bss_start
08049145 D _edata
08049148 B Numbers
08049170 B _end
```

Здесь внешние символьные имена представлены путем указания их типа прописной буквой (T, D или B).

Адресные значения внешних символьных имен, помещенные в команды ЯКЦП редактором связей

Команда дизассемблирования:

```
>objdump -d main > disas.main.txt
```

выводит:

```
main:     формат файла elf32-i386


Дизассемблирование раздела .text:

08048074 <_start>:
 8048074:    90                      nop
 8048075:    a1 25 91 04 08          mov    0x8049125,%eax
 804807a:    89 04 24                mov    %eax,(%esp)
 804807d:    a1 29 91 04 08          mov    0x8049129,%eax
 8048082:    8b 3d 2d 91 04 08       mov    0x804912d,%edi
 8048088:    8b 35 31 91 04 08       mov    0x8049131,%esi EBPm4
 804808e:    60                      pusha
 804808f:    68 17 91 04 08          push   $0x8049117
 8048094:    6a 08                   push   $0x8
 8048096:    e8 37 00 00 00          call   80480d2 <Read_Sym>
 804809b:    83 c4 08                add    $0x8,%esp
 804809e:    61                      popa
 804809f:    bb 00 00 00 00          mov    $0x0,%ebx
 80480a4:    b8 01 00 00 00          mov    $0x1,%eax
 80480a9:    cd 80                   int    $0x80

080480ab <Trans_Sym>:
 80480ab:    55                      push   %ebp
 80480ac:    89 e5                   mov    %esp,%ebp
 80480ae:    83 ec 04                sub    $0x4,%esp
 80480b1:    a1 35 91 04 08          mov    0x8049135,%eax
 80480b6:    89 45 fc                mov    %eax,-0x4(%ebp)
 80480b9:    8b 45 08                mov    0x8(%ebp),%eax
 80480bc:    3c 39                   cmp    $0x39,%al
 80480be:    77 09                   ja     80480c9 <Ret_error>
 80480c0:    3c 30                   cmp    $0x30,%al
 80480c2:    72 05                   jb     80480c9 <Ret_error>
 80480c4:    83 e8 30                sub    $0x30,%eax
 80480c7:    eb 05                   jmp    80480ce <Ret_norm>

080480c9 <Ret_error>:
```

```
 80480c9:    b8 ff ff ff ff       mov    $0xffffffff,%eax

080480ce <Ret_norm>:
 80480ce:    89 ec                mov    %ebp,%esp
 80480d0:    5d                   pop    %ebp
 80480d1:    c3                   ret

080480d2 <Read_Sym>:
 80480d2:    55                   push   %ebp
 80480d3:    89 e5                mov    %esp,%ebp
 80480d5:    83 ec 04             sub    $0x4,%esp
 80480d8:    a1 39 91 04 08       mov    0x8049139,%eax
 80480dd:    89 45 fc             mov    %eax,-0x4(%ebp)
 80480e0:    29 c9                sub    %ecx,%ecx

080480e2 <NextSym>:
 80480e2:    8b 55 0c             mov    0xc(%ebp),%edx
 80480e5:    29 db                sub    %ebx,%ebx
 80480e7:    8a 1c 0a             mov    (%edx,%ecx,1),%bl
 80480ea:    a1 3d 91 04 08       mov    0x804913d,%eax
 80480ef:    8b 3d 41 91 04 08    mov    0x8049141,%edi
 80480f5:    8b 35 31 91 04 08    mov    0x8049131,%esi EBPm4
 80480fb:    60                   pusha
 80480fc:    53                   push   %ebx
 80480fd:    e8 a9 ff ff ff       call   80480ab <Trans_Sym>
 8048102:    83 c4 04             add    $0x4,%esp
 8048105:    89 04 8d 48 91 04 08 mov    %eax,0x8049148(,%ecx,4)
                                         Numbers
 804810c:    61                   popa
 804810d:    41                   inc    %ecx
 804810e:    3b 4d 08             cmp    0x8(%ebp),%ecx
 8048111:    75 cf                jne    80480e2 <NextSym>
 8048113:    89 ec                mov    %ebp,%esp
 8048115:    5d                   pop    %ebp
 8048116:    c3                   ret
```

В этом файле в полях «Смещение» команд, содержащих внешние имена их значения показаны жирным шрифтом и приписаны соответствующие им символьные имена. Видно пokное соответствие этих значений с данными выше в выводе команды `nm -n main`.

Утилита `make`

Автор Стюарт Фельдман (Stuart Feldman), 1977 г., Bell Labs.

Утилита `make`, в частности, автоматически определяет какие исходные модули большой программы должны быть перетранслированы, и выполняет необходимые для этого действия.

По сути make это генератор shell скриптов сборки результирующего файла (ов) из исходных. До разработки make для сборки использовались именно shell скрипты.

Вы можете использовать make с любым языком программирования для которого имеется компилятор, работающий из командной строки. На самом деле, область применения make не ограничивается только сборкой программ. Можно использовать ее для решения любых задач, где одни файлы должны автоматически обновляться при изменении других файлов. Например можно собирать отчет в формате pdf из файлов на языке TeX и рисунков в форматах pdf и jpg.

Перед тем, как использовать make, нужно создать так называемый Makefile (это имя по умолчанию, можно использовать другое имя), который будет описывать зависимости между файлами вашей программы, и содержать команды для обновления целевых файлов.

В нашем случае исполняемый файл программы зависит от объектных файлов, которые, в свою очередь, зависят от файлов исходных текстов и получаются в результате их компиляции (а нашем случае ассемблирования).

После того, как Makefile создан команды make будет достаточно для выполнения всех необходимых перетрансляций если какие-либо из исходных файлов программы были изменены. Используя информацию из Makefile и зная время последней модификации файлов, утилита make решает, каких из файлов должны быть обновлены. Для каждого из этих файлов будут выполнены указанные в Makefile команды.

Простейший Makefile состоит из синтаксических конструкций всего двух типов: целей и макроопределений (сейчас не рассматриваем).

Цель в Makefile - это файл(ы), построение которого предполагается в процессе компиляции проекта. Описание цели состоит из трех частей:

- **имя цели**
- **список зависимостей**
- **список команд интерпретатора shell, требуемых для построения цели.**

Имя цели - непустой список файлов, которые предполагается создать. Список зависимостей - список файлов, из которых строится цель. Имя цели и список зависимостей составляют заголовок цели, записываются в одну строку и разделяются двоеточием. Список команд записывается со следующей строки, причем все команды начинаются с обязательного символа табуляции.

Возможна многострочная запись заголовка или команд через применение символа "\" для экранирования конца строки. При вызове команды make, если ее аргументом явно не указана цель, будет обрабатываться первая найденная в Makefile цель, имя которой не начинается с символа ".".

Рассмотрим Makefile из примера «Пример раздельной трансляции» с веб стр. нашего курса.

```
main: main.o Trans_Sym.o Read_Sym.o
        ld -melf_i386 -o main main.o Trans_Sym.o
Read_Sym.o
main.o: main.S my-macro
```

```
        as -ahlsm=main.lst --32 -gstabs+ -o main.o
main.S
Trans_Sym.o: Trans_Sym.S
        as -ahlsm=Trans_Sym.lst --32 -gstabs+ -o
Trans_Sym.o Trans_Sym.S
Read_Sym.o: Read_Sym.S
        as -ahlsm=Read_Sym.lst --32 -gstabs+ -o
Read_Sym.o Read_Sym.S
```

Итак, для каждого файла, который мы должны получить в процессе компиляции в файле `Makefile` указано, на основе каких файлов и с помощью какой команды он создается.

Утилита `make`, во первых собирает из этой информации правильную последовательность команд `shell` для получения требуемых результирующих файлов и, во-вторых, инициирует создание требуемого файла только в случае, если такого файла не существует, или он создан раньше, чем файлы от которых он зависит.

Таким образом в примере раздельной трансляции файл `make` будет ассемблировать только те модули, в которых были исправлены ошибки.

NB. Команда `make -B` получит все цели без проверки условий.

NBNB. Итак `make` эффективно поддерживает технологию раздельной трансляции и сборки с помощью редактора связей на основе понятия объектного файла.

Преимущества раздельной трансляции:

- **Поддержка модульного подхода (небольшие легко отлаживаемые**

модули).

- **Часто используемые модули можно объединить в стандартные библиотеки.**
- **В командной строке запуска редактора связей ld можно, указав имя библиотек, заставить редактор связей искать недостающие модули в этих библиотеках.**
- **Формат объектного файла является стандартом операционной системы, т.е. исполняемый модуль можно собирать из объектных модулей, полученных на разных языках –** ассемблер, C , fortran, **и т.д.**