ЭВМ и Периферийные устройства

лекция 8

Структуры

Структура это набор переменных (данных). Структура задаётся с помощью директивы struct и ends. Перед использованием структуры её нужно описать:

```
SOMESTRUCTURE STRUCT
dword1 dd ?
dword2 dd ?
some_word dw ?
abyte db ?
anotherbyte db ?
SOMESTRUCTURE ENDS
```

Уже после можно объявлять её конкретные экземпляры.

Структуры

Структуры можно объявлять как в секции .data, так и в секции .data?

```
MYSTRUCT struc
dword1 dd ?
dword2 dd ?
some_word dw ?
abyte db ?
anotherbyte db ?
MYSTRUCT ends

.data
msg1  MYSTRUCT <?>
.data?
msg2  MYSTRUCT <?>
```

Структуры

```
MYSTRUCT struc
dword1 dd ?
dword2 dd ?
some_word dw ?
abyte db ?
anotherbyte db ?
MYSTRUCT ends
```

Для того чтобы получить доступ к записи надо указать метку переменной, которой она обозначена и через точку указать имя поля.

```
mov [msg.dword1], 45h
xor eax,eax
mov  eax, [msg.dword1] ; eax = 45
```

при этом запись msg.dword1 считается обычной меткой данных: берётся смещение метки msg плюс смещение поля dword1 в структуре, размер данных по умолчанию равен размеру директивы указанной после метки поля. Также можно пользоваться обращением к полю при обращении к записи через регистр:

```
mov [msg.dword2], 45h
xor eax,eax
lea  ebx, msg
mov  eax, [ebx].dword2 ; eax = 45
```

Структуры

```
MYSTRUCT struc
dword1 dd ?
dword2 dd ?
some_word dw ?
abyte db ?
anotherbyte db ?
MYSTRUCT ends
```

Если имя поля не гарантирует уникальности то лучше использовать такой тип использования записи:

```
mov [msg.dword2], 45h
xor eax,eax
lea  ebx, msg
mov  eax, [ebx].MYSTRUCT.dword2 ; eax = 45
```

Указанная запись гарантирует, что мы получаем доступ к нужному нам полю, нужной нам структуры.

Структуры

```
MYSTRUCT struc
dword1 dd ?
dword2 dd ?
some_word dw ?
abyte db ?
anotherbyte db ?
MYSTRUCT ends
```

Как и всё остальное, структуры- это всего-лишь данные в памяти. Поэтому вместо обращения по конкретным именам, можно использовать смещение в памяти

```
mov [msg.abyte], 45h
xor eax,eax
lea  ebx, msg
mov  al, [ebx+10d] ; al = 45
```

к ebx прибавлено10d, потому что смещение поля abyte в структуре равно 10d.

Макрос chr$()

chr$(). Он создает в секции .data массив байт и передает указатель на них в функцию. Для юникода есть аналогичный макрос **uni$()**. В итоге, вызов функции приобретает такой вид:

```
invoke MessageBox,0,chr$("Text"),chr$("Caption"),MB_OK
```

Идентификатор $

$ всегда равен текущему смещению в сегменте, в котором в данный момент выполняет ассемблирование.
С помощью него можно получить размер чего-либо (переменной, массива, структуры)
Предположим, например, что вы хотите приравнять идентификатор STRING_LENGTH к длине строки в байтах.

```
; Без $ можно и так:
StringStart LABEL BYTE
  db 0dh,0ah,'Текстовая строка'odh,0ah
StringEnd LABEL BYTE
STRING_LENGTH EQU (StringEnd-StringStart)
```

```
; А с $ всё проще:
StringStart LABEL BYTE
db 0dh,0ah,'Текстовая строка'odh,0ah
STRING_LENGTH EQU ($-StringStart)
```

```
; Длину (в словах) массива слов можно вычислить следующим образом:
WordArray DW 90h, 25h, 0, 16h, 23h
WORD_ARRAY_LENGTH EQU (($-WordArray)/2)
```

Макрос SIZEOF

SIZEOF служит для определения размера чего-либо (переменной, массива, строки, структуры). Может применятся, как константа.

```
.data
str1  db   '...'
len_str1=sizeof str1; можно и так

…
start:
mov     cx,sizeof str1; и так
```

Строки

Строки представляются из себя группу байт, слов, двойных слой в памяти. Строки обычно объявляются в сегменте данных вот так:

```
.data
response db 20 DUP (?)
label1 BYTE 'The results are ', 0
wordString WORD 50 DUP (?)
arrayD DWORD 60 DUP (0)
```

Чисто технически строки и массивы – это одно и тоже. Просто строки мы воспринимаем не просто как массивы, а как строки. Рассмотренные ниже операции применимы и к обычным массивам.

Установка флага DF

Команда CLD сбрасывает флаг DF в 0.
Комманда STD устанавливает флаг DF в 1

Rep.

Rep – выполняет указанную команду, пока cx<>0 при этом уменьшая cx
Действия rep:
1) анализ содержимого cx:

- если cx<>0, то выполнить цепочечную команду, следующую за данным префиксом и перейти к шагу 2;
- если cx=0, то передать управление команде, следующей за данной цепочечной командой (выйти из цикла по rep);

2) уменьшить значение cx=cx–1 и вернуться к шагу 1;

Префикс Rep.

Rep – выполняет указанную команду, пока (e)cx<>0 при этом уменьшая (e)cx. rep используется перед следующими цепочечными командами и их краткими эквивалентами: movs, stos, ins, outs.

Действия rep:

1) анализ содержимого (e)cx:
   - если (e)cx<>0, то выполнить цепочечную команду, следующую за данным префиксом и перейти к шагу 2;
   - если (e)cx=0, то передать управление команде, следующей за данной цепочечной командой (выйти из цикла по rep);
2) уменьшить значение (e)cx=(e)cx–1 и вернуться к шагу 1;

Префиксы Repe и Repz.

epe и repz используются перед следующими цепочечными командами и их краткими эквивалентами: cmps, scas. Действия repe и repz:

1) анализ содержимого cx и флага zf:
   - если (e)cx<>0 (у нас ecx)или zf<>0, то выполнить цепочечную команду, следующую за данным префиксом, и перейти к шагу 2;
   - если (e)cx=0 (у нас ecx) или zf=0, то передать управление команде, следующей за данной цепочечной командой (выйти из цикла по rep);
2) уменьшить значение (e)cx=(e)cx-1 и вернуться к шагу 1;

Префиксы Repe и Repz.

repe и repz используются перед следующими цепочечными командами и их краткими эквивалентами: cmps, scas. Действия repe и repz:

1) анализ содержимого cx и флага zf:
   - если (e)cx<>0 (у нас ecx) или zf<>0, то выполнить цепочечную команду, следующую за данным префиксом, и перейти к шагу 2;
   - если (e)cx=0 (у нас ecx) или zf=0, то передать управление команде, следующей за данной цепочечной командой (выйти из цикла по rep);
2) уменьшить значение (e)cx=(e)cx-1 и вернуться к шагу 1;

Префиксы Repne и Repnz.

epne и repnz также имеют один код операции и имеют смысл при использовании перед следующими цепочечными командами и их краткими эквивалентами: cmps, scas. Действия repne и repnz:

1)анализ содержимого (e)cx и флага zf:

- если (e)cx<>0 (у нас ecx) или zf=0, то выполнить цепочечную команду, следующую за данным префиксом и перейти к шагу 2;
- если (e)cx=0 (у нас ecx) или zf<>0, то передать управление команде, следующей за данной цепочечной командой (выйти из цикла по rep);

2)уменьшить значение (e)cx=(e)cx–1 и вернуться к шагу 1.

Строки. Команды для работы со строками.

Как уже упоминалось, со строками можно работать как с массивами. Однако есть специальные команды, упрощающие работу:

LODS - load string, загрузить строку
STOS - store string, сохранить строку
MOVS - move string, переместить строку
CMPS - compare string, сравнить строку
SCAS - scan string, сканировать строку

Строки. Использование команд для работы со строками.

Все указанные инструкции можно использовать как с параметрами, так и без.

Если вы используете указанные инструкции без параметров, то вам нужно дописывать одну букву к команде, чтобы указать размер элемента (символа) строки, с которой вы работаете.

Например у LODS это выглядит так:

LODS**B** – для работы с байтовыми операндами (**b**yte)
LODS**W** – для работы с двухбайтовыми операндами (словами, **w**ord)
LODS**D** – для работы с четырёхбайтовыми операндами (двойное слово, **d**ouble word)

Тот же LODS можно вызывать с параметрами, например так:

```
LODS byteVar
```

В этом случае LODS поймет, что работает с однобайтовыми символами по размеру byteVar. **Важно! При этом значение и смещение byteVar не влияют на исполнение команды. byteVar нужен просто для определения размера.**

Если же параметр не указывается – то требуются мнемоники B, W или D, дописанные к команде.

LODS

(LOaD String Byte/Word/Double word operands)
Загрузка строки байтов/слов/двойных слов

LODS загружает символ из памяти по адресу esi в al/ax/eax и смещает значение esi.

```
lods источник
lodsb
lodsw
lodsd
```

- Загрузить элемент из ячейки памяти, адресуемой парой ds:esi/si (для нас esi), в регистр al/ax/eax. Размер элемента определяется неявно (для команды lods) или явно в соответствии с применяемой командой (для команд lodsb, lodsw, lodsd);
- Изменить значение регистра esi на величину, равную длине элемента цепочки. Знак этой величины зависит от состояния флага направления df:
  - df=0 — значение положительное, то есть просмотр от начала цепочки к ее концу;
  - df=1 — значение отрицательное, то есть просмотр от конца цепочки к ее началу.

LODS. Пример.

```
str     dw      ...
...
        cld
        lea     si,str
        lodsw   ;загрузить первые 2 байта из str в ax.
                 ; При этом esi увеличилась на 2, так как флаг df=0
```

```
;загрузить первые 2 байта из str в ax можно и так:
mov esi,0
mov ax,[esi]
inc esi
inc esi
;Но первый пример работает быстрее. Да и удобнее.
```

Обычно команду LODS используют в некотором цикле для просмотра некоторой цепочки с элементами фиксированного размера.

STOS

(Store String Byte/Word/Double word operands)
Сохранение строки байтов/слов/двойных слов

LODS наоборот. STOS сохраняет символ из al/ax/eax в память по адресу edi и смещает значение в edi.

```
stos приемник
stosb
stosw
stosd
```

- Записать элемент из регистра al/ax/eax в ячейку памяти, адресуемую парой es:di/edi (у нас edi). Размер элемента определяется неявно (для команды stos) или конкретной применяемой командой (для команд stosb, stosw, stosd);
- Изменить значение регистра di на величину, равную длине элемента цепочки. Знак этого изменения зависит от состояния флага df:
  - df=0 — значение положительное, то есть просмотр от начала цепочки к ее концу;
  - df=1 — значение отрицательное, то есть просмотр от конца цепочки к ее началу.

STOS. Пример.

```
;заполнить некоторую область памяти пробелами.
.data
str1     db      'Какая-то строка'
...
  cld
        mov     al,' '
        lea     edi,str1
        mov     cx,sizeof str1
rep     stosb           ;заполняем пробелами строку strr
```

Пример совместной работы LODS и STOS

```
;пример совместной работы stosb и lodsb:
;копировать одну строку в другую до первого пробела
.data
str1    db      'Какая-то строка'
len_str1=sizeof str1
str2    db      len_str1 dup (' ')
...
        cld
        mov     cx,len_str1
        lea     esi,str1
        lea     edi,str2
m1:     lodsb
        cmp     al,' '
        je      exit    ;выход, если пробел
        stosb
        loop    m1
exit:
```

MOVS

(MOVe String Byte/Word/Double word)
Пересылка строк байтов/слов/двойных слов

LODS + STOS одной командой. Скопировать символ из адреса в esi по адресу в edi и сместить значение в esi и edi.

```
movs приемник,источник
movsb
movsw
movsd
```

•Выполнить копирование байта, слова или двойного слова из операнда источника в операнд приемник, при этом адреса элементов предварительно должны быть загружены:

- адрес источника — в пару регистров ds:esi/si (у нас esi) (*ds* по умолчанию, допускается замена сегмента);
- адрес приемника — в пару регистров es:edi/di (у нас edi) (замена сегмента не допускается);

•в зависимости от состояния флага df изменить значение регистров esi/si и edi/di:

- если df=0, то увеличить содержимое этих регистров на длину структурного элемента последовательности;
- если df=1, то уменьшить содержимое этих регистров на длину структурного элемента последовательности;

•если есть префикс повторения, то выполнить определяемые им действия (см. команду rep).

MOVS. Пример.

```
;копировать одну строку в другую полностью
str1    db      'str1 копируется в str2'
len_str1=sizeof str1
a_str1  dd      str1
str2    db      len_str1 dup (' ')
a_str2  dd      str2
...
        mov     cx,len_str1
        lea     si,str1
        lea     di,str2
        cld
rep     movsb
```

CMPS

(CoMPare String Byte/Word/Double word operands)
Сравнение строк байтов/слов/двойных слов

Сравнение двух последовательностей (цепочек) элементов в памяти по адресам esi и edi.

```
cmps приемник,источник
cmpsb
cmpsw
cmpsd
```

•выполнить вычитание элементов (источник - приемник), адреса элементов предварительно должны быть загружены:

- адрес источника — в пару регистров *ds*:esi/si (у нас esi);
- адрес назначения — в пару регистров es:edi/di (у нас edi);

•в зависимости от состояния флага df изменить значение регистров esi/si и edi/di:

- если df=0, то увеличить содержимое этих регистров на длину элемента последовательности;
- если df=1, то уменьшить содержимое этих регистров на длину элемента последовательности;

•в зависимости от результата вычитания установить флаги:

- если очередные элементы цепочек не равны, то cf=1, zf=0;
- если очередные элементы цепочек или цепочки в целом равны, то cf=0, zf=1;

•при наличии префикса выполнить определяемые им действия (см. команды repe/repne).

CMPS. Пример.

```
.data
obl1 db 'Строка для сравнения'
obl2 db 'Строка для сравнения'
.code
...
 cld ;просмотр цепочки в направлении возрастания адресов
 mov cx,20 ;длина цепочки
 lea esi,obl1 ;адрес источника поместить esi
 lea edi,obl2 ;адрес назначения поместить в edi
repe cmpsb ;сравнивать, пока равны
 jnz m1 ;если не конец цепочки, то встретились разные элементы
... ;действия, если цепочки совпали
...
m1:
... ;действия, если цепочки не совпали
```

SCAS

(SCAn String Byte/Word/Double word operands)
Сканирование строки байтов/слов/двойных слов

Поиск значения в последовательности (цепочке) элементов в памяти.

```
scas приемник
scasb
scasw
scasd
```

•выполнить вычитание (элемент цепочки-(eax/ax/al)). Элемент цепочки локализуется парой es:edi/di (у нас edi). Замена сегмента es не допускается;

•по результату вычитания установить флаги;

•изменить значение регистра edi/di (у нас edi) на величину, равную длине элемента цепочки. Знак этой величины зависит от состояния флага df:

- df=0 — величина положительная, то есть просмотр от начала цепочки к ее концу;
- df=1 — величина отрицательная, то есть просмотр от конца цепочки к ее началу.

SCAS. Пример.

```
;сосчитать число пробелов в строке str
.data
str1  db   '...'
len_str1=sizeof str1
.code
lea  edi,str1
  mov  cx,len_str1    ;длину строки — в cx
  mov  al,' '
  mov  bx,0 ;счетчик для подсчета пробелов в строке
  cld
cycl:
repne scasb
  jcxz exit ;переход на exit, если цепочка просмотрена полностью
  inc  bx
  jmp  cycl
exit:   ...
```