Подпрограммы

Использование подпрограмм

Подпрограмма (процедура) - набор взаимосвязанных команд, обычно выполняющих одну часто встречающуюся операцию.

Требования к подпрограммам:

- Выполняет одну и только одну работу.
- Коротка, насколько это возможно, и велика, насколько это необходимо.
- Начинается не с кода, а с комментариев, описывающих ее назначение, результаты, входные данные и используемые регистры.

• Директивы **PROC** и **ENDP** отмечают начало и окончание подпрограммы.

• Команды **call** и **ret** вызывают подпрограмму и выходят из нее соответственно.

Директива PROC определяет имя подпрограммы, помечающее адрес первой ее команды

Директива ENDP завершает подпрограмму, за ней следует имя той же метки, что и в директиве PROC.

Вызов подпрограмм

команда	адрес	
call адрес	регистр переменная число (метка)	2 или 4 байта, абсолютный или относительный (смещение со знаком)
call far адрес вызов из другого сегмента	переменная число	4 или 6 байт: 2 (байта селектор сегмента) + 2 или 4 байта – эфф. адрес

Вызов подпрограмм

- **call** выполняет:
  - помещает в стек адрес следующей за call команды (это адрес возврата)
  - EIP := адрес
- **call far** дополнительно
  - проверяет привилегии
  - помещает в стек CS (до адреса возврата)
  - меняет CS

Возврат из подпрограммы

команда	
ret [число] **retn** [число]	
retf [число]	из дальнего вызова

- [число] – необязательный параметр, используется для очистки стека от аргументов

Возврат из подпрограммы

- **ret**/**retn** выполняет:
  - извлекает из стека EIP
  - если задано [число], то после извлечения EIP – извлекает из стека еще [число] байт
- **retf** дополнительно:
  - извлекает из стека CS (сразу после EIP)

Шаблон оформления подпрограммы

```
Start: ...
……..
        call    AddRegisters     ; Вызов п/п
……..
AddRegisters PROC
        ....
        ret             ; Возврат в вызывающую программу
AddRegisters ENDP
…
END     Start           ; Конец программы
```

Декларация подпрограмм

```
.code
incr proc     ; это метка – имя п/п!
    inc ax
    ret
incr endp
main proc          ; и это метка
    ...
    call incr
    ...
main endp
end main ; метка, с которой начнется
         ; выполнение программы
```

Средства ассемблера

- Директива **proc** имеет необязательные параметры, упрощающие написание подпрограмм
- В MASM есть макроопределения для прототипов процедур **proto** и их вызова **invoke**
- Т.к. цель данного курса – изучение не возможностей компилятора, а архитектуры компьютера, они не рассматриваются и их использование в лабораторных запрещено

Передача значений в подпрограммы и из них

1. Передача значений в подпрограмму через регистры является наиболее общим методом получения подпрограммами данных для обработки.
2. Запись данных в глобальные переменные
3. Передача данных через стек*

Желательно – РОН, которые «портит» процедура, сохранить перед вызовом процедуры и восстановить после*.

ПА через регистры

- Наиболее быстрый способ
- Для малого числа аргументов
- Вызывает трудности с компиляцией. В языках высокого уровня применяется обычно только для возврата результата

- Пример – см. *incr* выше

ПА через глобальные переменные

- И вызывающая, и вызываемая процедура знают абсолютный адрес переменных
- По скорости уступает только ПА через регистры
- Невозможна рекурсия и многопоточность, ограничена масштабируемость, источник ошибок
- Не применяйте ни на Assembler, ни на языках высокого уровня!

ПА через переменные по адресу (указателю)

- Процедуре передается адрес области памяти:
  - через регистр
  - через стек
  - через глобальную переменную

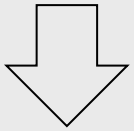
  } одним из остальных способов
- Процедура считывает из памяти значения и записывает туда результат
- Применяется для передачи
  - данных большого объема
  - выходных аргументов

ПА через стек

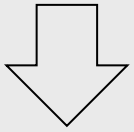
- Вызывающий помещает в стек
  - сначала параметры
  - затем EIP (командой **call**)
- Вызываемая процедура
  - не извлекает параметры с помощью **pop** (EIP в стеке будет использовано в **ret**)
  - *может* обращаться к параметрам через [ESP+смещение]
  - обычно выполняет EBP := ESP и обращается через [EBP+смещение], т.к. внутри процедуры ESP обычно меняется

Сохранение регистров в стеке

- Вызывающая процедура может хранить значения в регистрах до вызова
- Вызываемая процедура "не знает", какие регистры использует вызывающая



- После возврата из процедуры значения регистров не должны быть нарушены



- Внутри вызова их нужно сохранять!

Сохранение регистров в стеке

команда	поместить в стек регистры общего назначения	вершина стека
pushaw	16-ти битные (AX CX DX BX SP BP SI DI)	SP := SP - 16
pusha **pushad**	32-х битные (EAX ECX EDX EBX ESP EBP ESI EDI)	ESP := ESP - 32

- pusha от push All
- Помещается значение ESP, которое было до начала работы команды

Сохранение регистров в стеке

команда	извлечь из стека регистры общего назначения
popaw	16-ти битные (DI SI BP ~~SP~~ BX DX CX AX)
popad	32-х битные (EDI ESI EBP ~~ESP~~ EBX EDX ECX EAX)

- Извлекает регистры в обратном порядке
- Значение ESP игнорирует, т.е. не меняет стек!