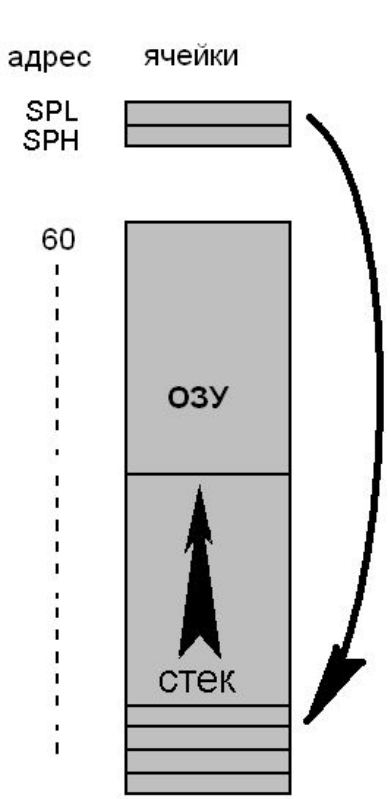
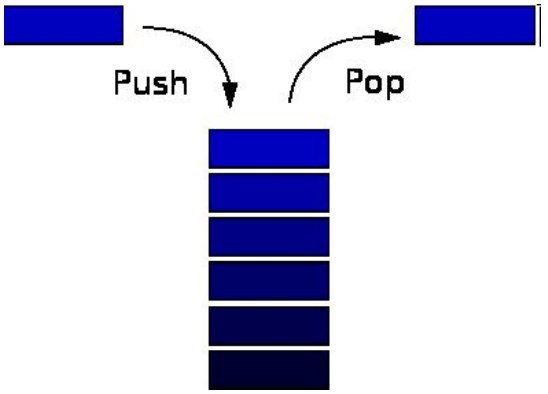
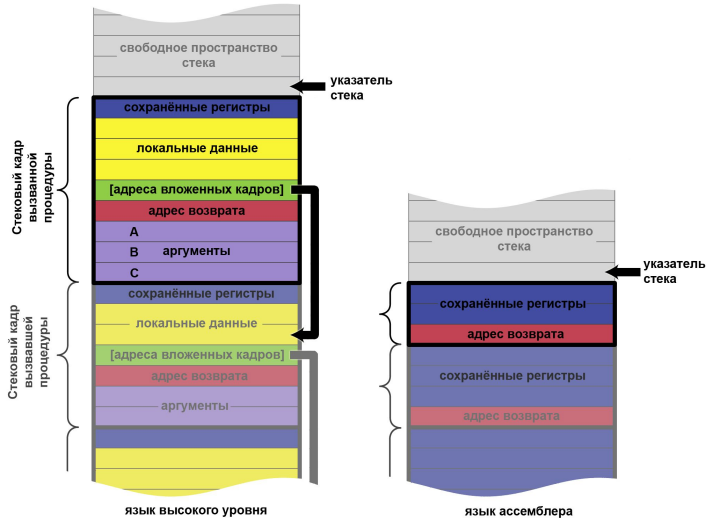
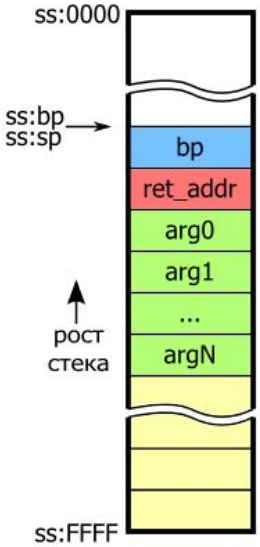
Lection №8

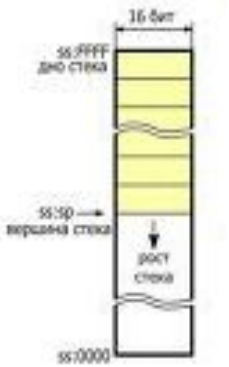
Стек. Приоритеты. Форма ПОЛИЗ.











Стек. Идеология

Во всех языках программирования есть понятие процедур с локальными переменными. Эти переменные доступны во время выполнения процедуры, **но!** перестают быть доступными после окончания процедуры.

?

Возникает вопрос: где должны храниться такие переменные?

К сожалению, предоставить каждой переменной абсолютный адрес в памяти невозможно. Проблема заключается в том, что процедура может вызывать себя сама.

ПРОБЛЕМА

если процедура вызывается дважды, то хранить ее переменные под конкретными адресами в памяти **нельзя**,
поскольку второй вызов нарушит результаты первого.

Что делать?

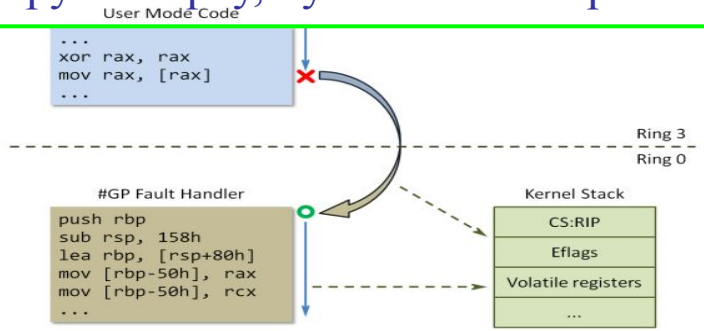
Работать с особой областью памяти: стеком

Стек. Идеология

Стек (англ. stack — стопка) — структура данных с методом доступа к элементам LIFO (англ. Last In — First Out, «последним пришёл — первым вышел»).

WIKIPEDIA

Чаще всего принцип работы стека сравнивают со стопкой тарелок: чтобы взять вторую сверху, нужно снять верхнюю.

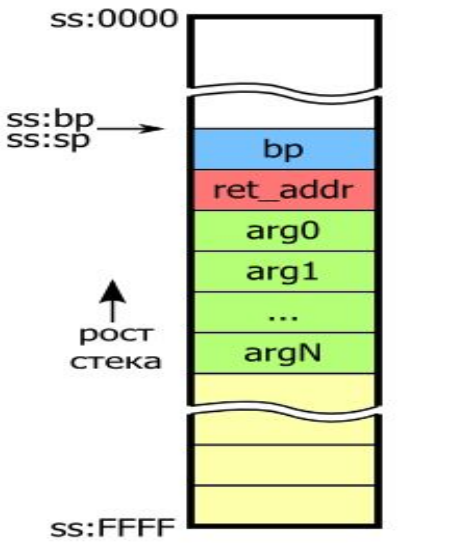


Важно!

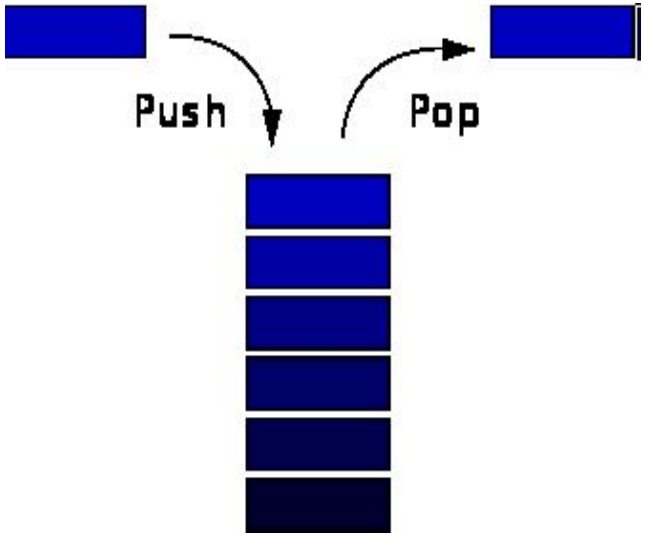
Важно отменить, что отдельные переменные в стеке не получают в нем абсолютных адресов.

Стек. Общие положения

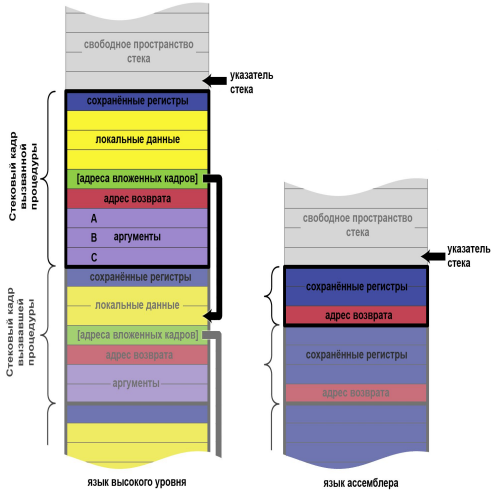
- Стек представляет собой непрерывную область памяти, адресуемую регистрами ESP (указатель стека) и SS (селектор сегмента стека)



- Данные помещаются в стек с помощью инструкций **PUSH**(заталкивание),а извлекаются по инструкции **POP**(вытаскивание)



- За одну операцию можно поместить или извлечь только слово или двойное слово(2 байта/4байта)



Стек. Принцип работы. Явный доступ.

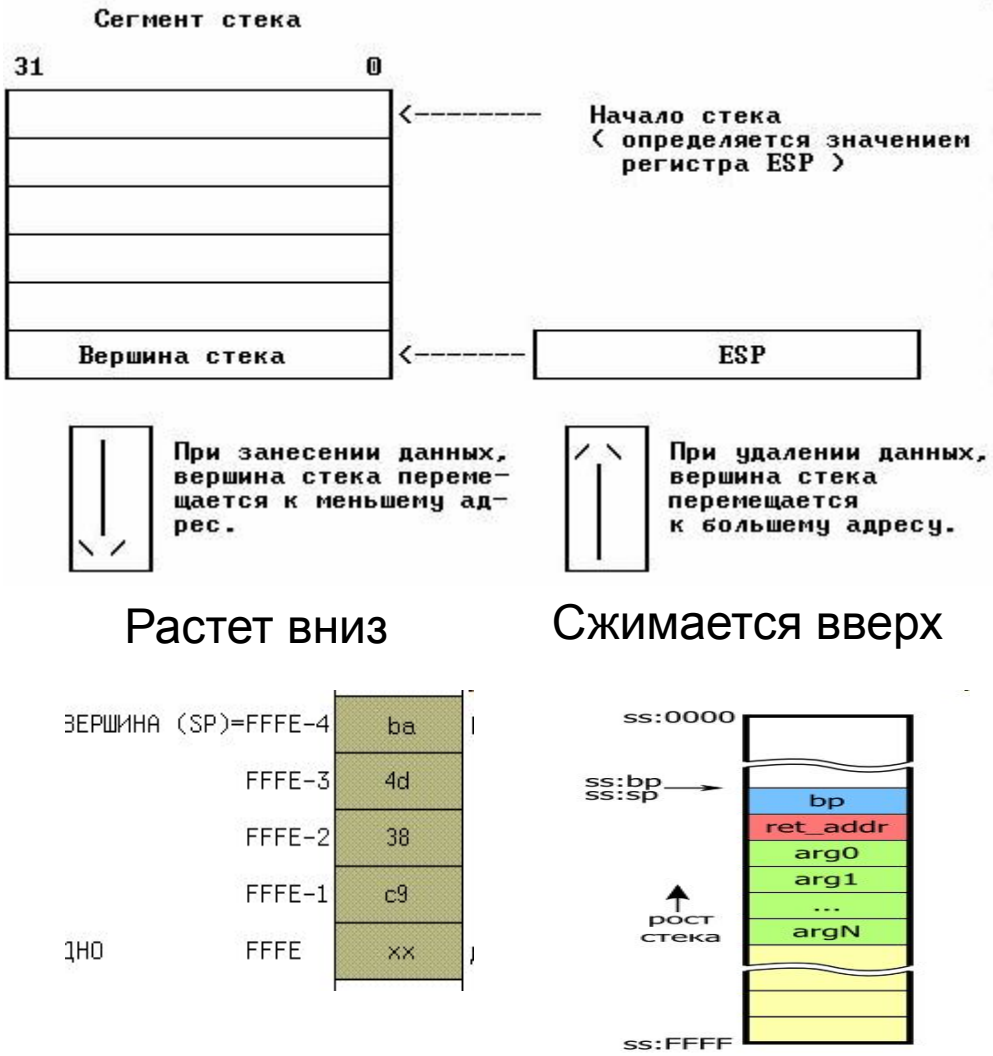
Указатель стека ESP (или SP) показывает на верхушку стека.

Инструкция PUSH:

- При помещении данных сначала декрементируется указатель стека на 2 или 4, в зависимости от разрядности данных.
- После этого, данные помещаются в сегмент , определённый регистром SS, со смещением, определяемым новым значением ESP.

Инструкция POP:

- При извлечении данные считываются из памяти по адресу SS:ESP, после чего указатель стека инкрементируется на 2 или 4.



Стек. Принцип работы. Альтернативный доступ.

Стек автоматически используется процессором при выполнении инструкций вызова (CALL), возврата(RET и IRET), входа(ENTER) и выхода(LEAVE) из процедур, а также при обработке прерываний.

PUSH-аналог

Инструкция CALL(инструкция вызова):

- В стек помещается адрес возврата – значения регистров CS и EIP, указывающие на инструкцию, следующую после инструкции вызова
- При ближайших вызовах сегментный регистр в стеке **не сохраняется**.

По прерываниям:

- В стек помещаются значения регистра EFLAGS,а затем адрес инструкции, следующей за той, на которой произошло прерывание.

POP-аналог

Инструкция RET :

- Эти значения извлекаются из стека в соответствующие регистры, и процессор продолжает выполнять прерванную последовательность инструкций.

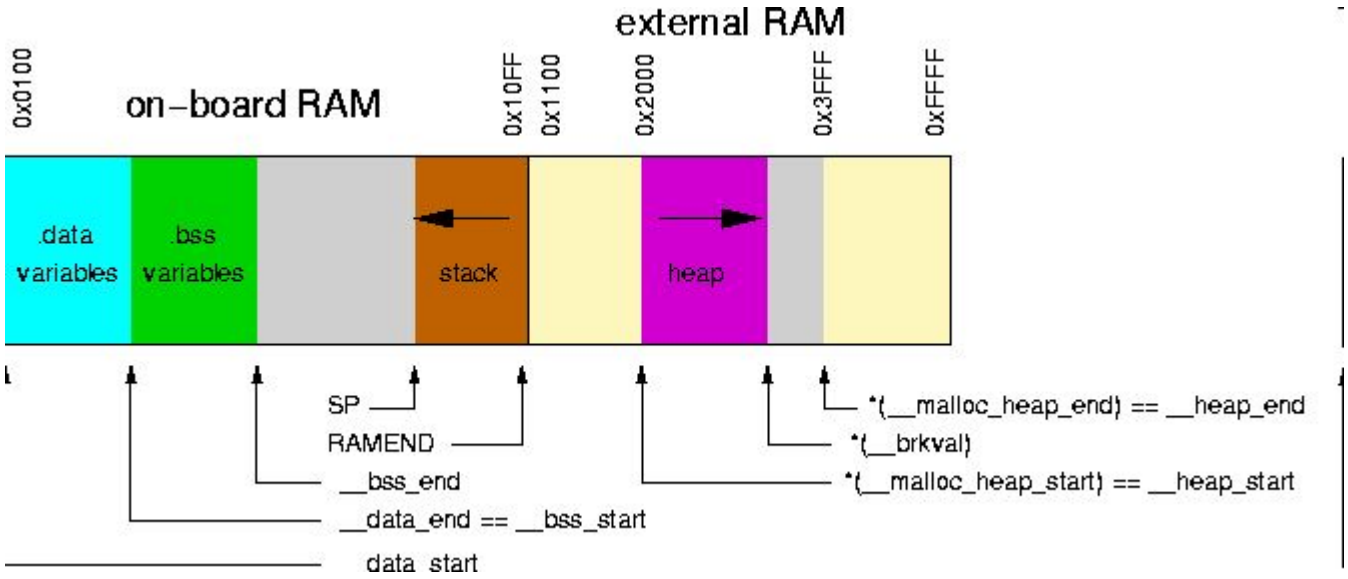
Инструкция IRET:

- Исключение работает аналогично прерыванию, но следом за содержимым EIP в ряде случаев в стек помещается также слово кода ошибки, которое должно быть извлечено обработчиком исключений.

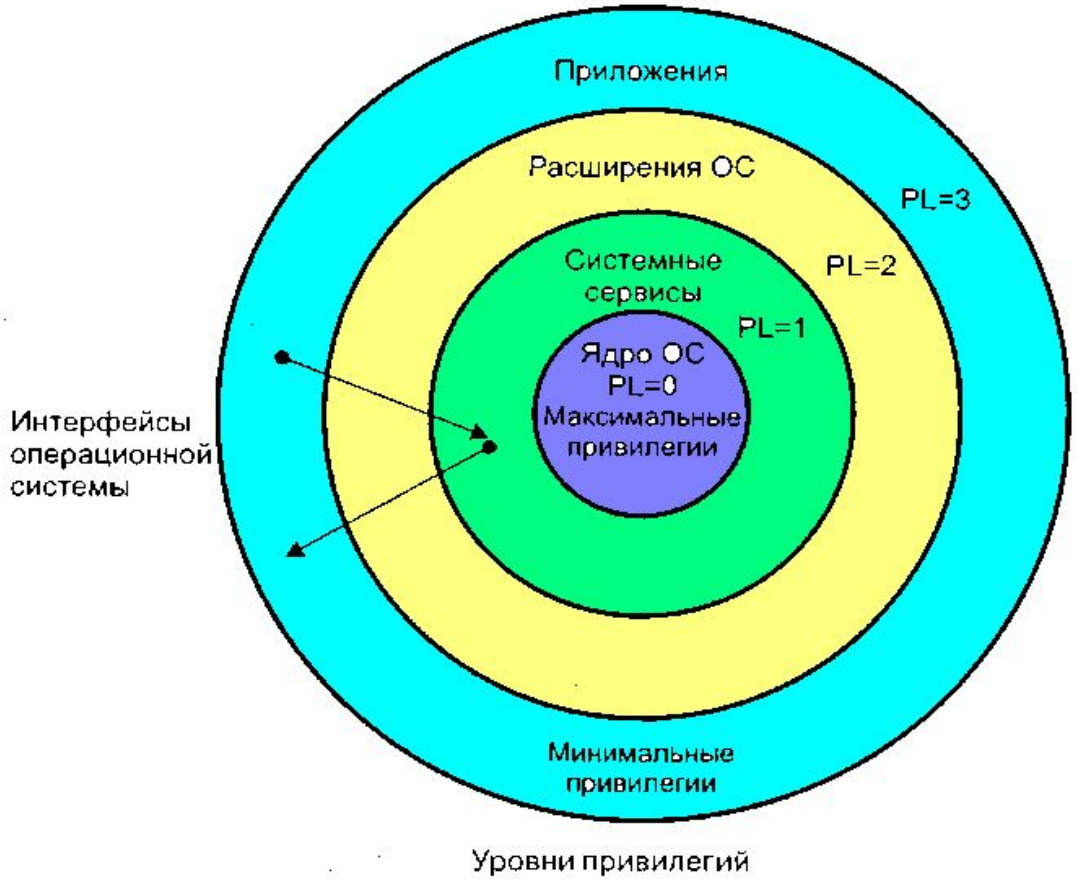
Стек. Назначение

Стек используют для различных целей:

- Организации прерываний, вызовов и возвратов;
- Временного хранения данных, когда под нет смысла выделять фиксированного места в памяти;
- Передачи возвратов параметров при вызовах процедур.



Приоритеты(привилегии)



PL = 0 – ядро
PL = 1 – системные сервисы
PL = 2 – Расширения ОС
PL = 3 - Приложения

Уровни привилегий относятся к дескрипторам, селекторам и задачам. Кроме того, в регистре флагов имеется *поле привилегий ввода-вывода*, с помощью которого обеспечивается доступ к инструкциям ввода-вывода и управлению флагом прерываний.

Форма ПОЛИЗ(обратная польская запись). Уровень архитектуры команд.

Обратная польская нотация (ОПН) (Обратная польская запись, Обратная бесскобочная запись (ОБЗ), Постфиксная нотация, Бесскобочная символика Лукашевича, Польская инверсная запись, ПОЛИЗ) — форма записи математических выражений, в которой операнды расположены перед знаками операций.

Jan Łukasiewicz;(Ян Лукашевич) 21 декабря 1878, Львов — 13 ноября 1956, Дублин) — польский логик, член Польской Академии Наук (1937), один из главных представителей львовско-варшавской школы.

Стековой машиной называется алгоритм, проводящий вычисления по обратной польской записи

Обратная польская запись совершенно унифицирована — она принципиально одинаково записывает унарные, бинарные, тернарные и любые другие операции, а также обращения к функциям, что позволяет не усложнять конструкцию вычислительных устройств при расширении набора поддерживаемых операций.

Инфиксная запись	Обратная польская запись
A+BxC	ABCx+
AxB+C	ABxC+
AxB+CxD	ABxCDx+
(A+B)/(C-D)	AB+CD-/
AxB/C	ABxC/
((A+B)xC+D)/(E+F+G)	A B+CxD+EF+G+/

Некоторые примеры инфиксных выражений и их эквиваленты в обратной польской записи

Вычисления на стеке. Стековая машина

Алгоритм вычисления для стековой машины

1. Обработка входного символа
- Если на вход подан операнд, он помещается на вершину стека.
- Если на вход подан знак операции, то соответствующая операция выполняется над требуемым количеством значений, извлечённых из стека, взятых в порядке добавления.

2. Результат выполненной операции кладётся на вершину стека.

Если входной набор символов обработан не полностью, перейти к шагу 1.

3. После полной обработки входного набора символов результат вычисления выражения лежит на вершине стека.

Пример вычисления выражений

Выражение ((1 + 2) * 4) + 3 в ОПН может быть записано так:

1 2 + 4 * 3 +

Вычисление производится следующим образом (указано состояние стека после выполнения операции):

Ввод	Операция	Стек
1	поместить в стек	1
2	поместить в стек	1,2
+	сложение	3
4	поместить в стек	3,4
*	умножение	12
3	поместить в стек	12,3
+	сложение	15

Результат, 15, в конце вычислений находится на вершине стека.

Использование стека процессором при обработке прерываний

процессор генерирует исключение, операционная система находит нужный обработчик и вызывает его.

Выполняя поток команд:

процессор проверят возможность выполнения каждой инструкции

корректность её аргументов

остальные факторы, влияющие на корректность выполнения кода

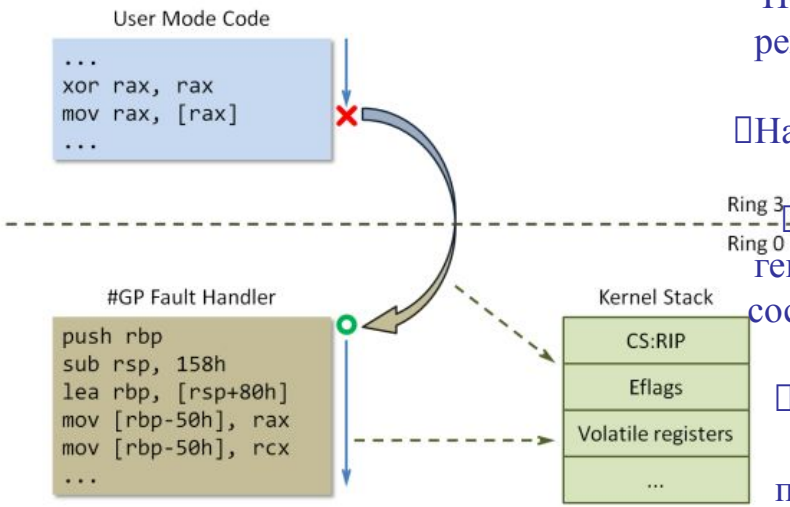
В случае если команда не может быть выполнена

- деление на ноль
- обращение к несуществующей странице
- несоответствие уровня привилегий
- Другое

процессор генерирует исключение – вызывает один из обработчиков, зарегистрированных операционной системой в IDT (Interrupt Dispatch Table).

Использование стека процессором при обработке прерываний

При вызове обработчика процессор делает сразу несколько вещей: переключается в режим ядра (Ring 0), переключает указатель стека на ядерный стек и сохраняет предыдущие указатели команд и стека в ядерном стеке.

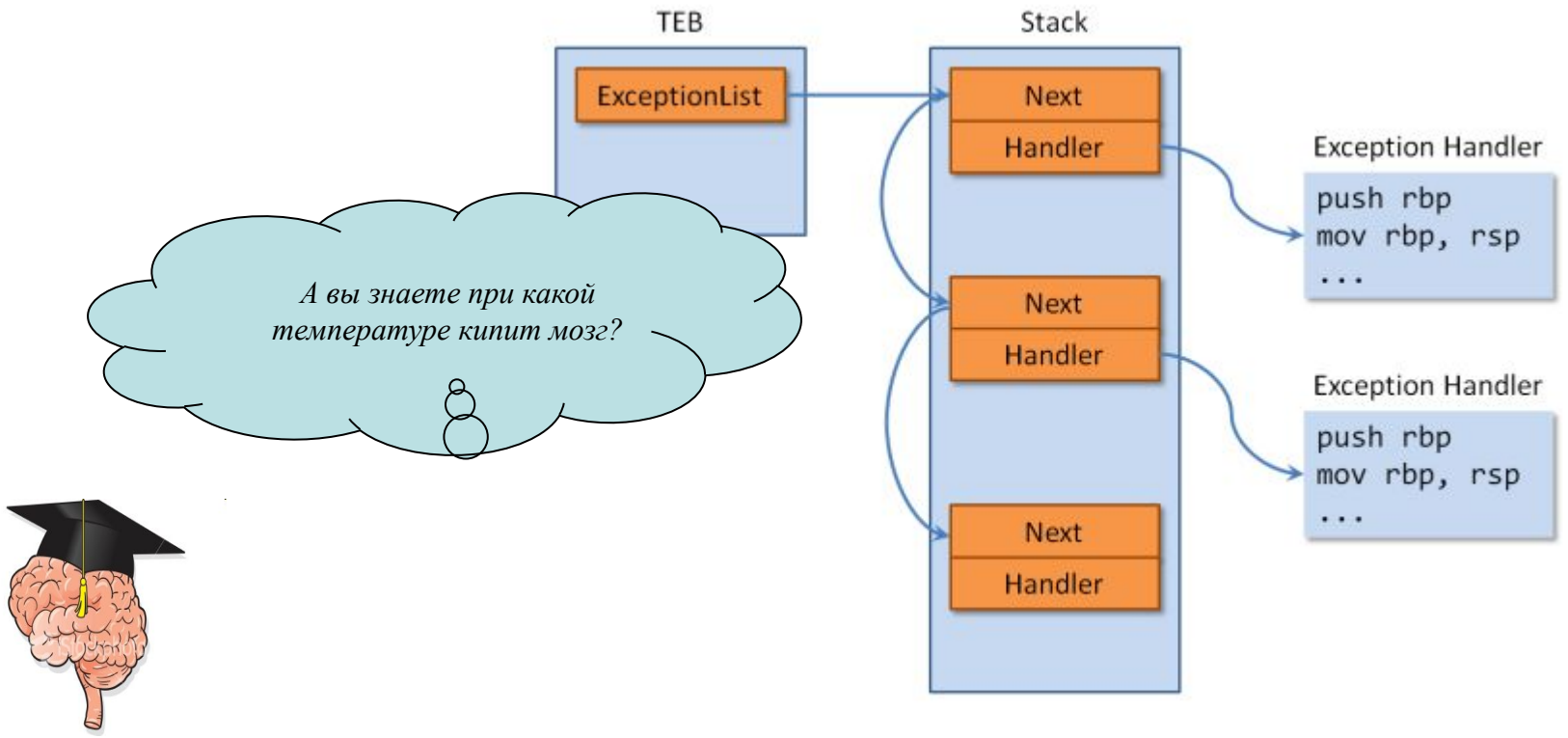


Получив контроль, обработчик исключения сохраняет остальные регистры процессора в стеке и выполняет действия, специфичные для конкретного исключения.

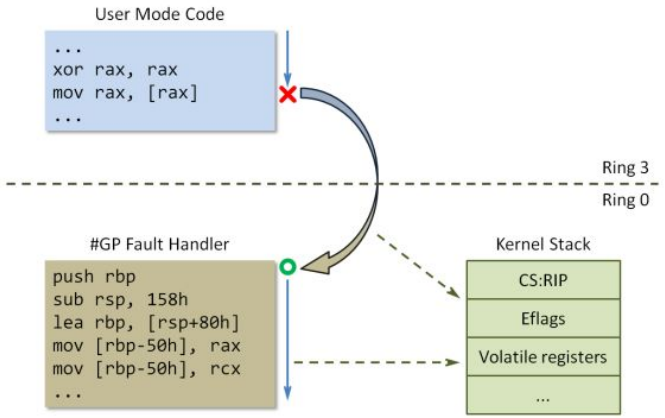
- Например, обработчик Page Fault Exception запрашивает подкачку страницы у Memory Manager.
- Если обработчику удалось разрешить проблему, вызвавшую генерацию исключения, обработчик восстанавливает сохраненное состояние процессора и выполняет возврат в пользовательский код (Ring 3).
- В противном случае, в дело вступает диспетчер исключений.
- Диспетчер исключений размещает структуру CONTEXT в пользовательском стеке и копирует туда сохранённое состояние регистров из ядерного стека.
- Туда же сохраняется текущее состояние Floating Point регистров.
- Информация об исключении записывается в структуру EXCEPTION_RECORD.
- Далее, диспетчер подменяет адрес возврата в пользовательский код адресом диспетчера исключений пользовательского режима и выполняет возврат в Ring 3.

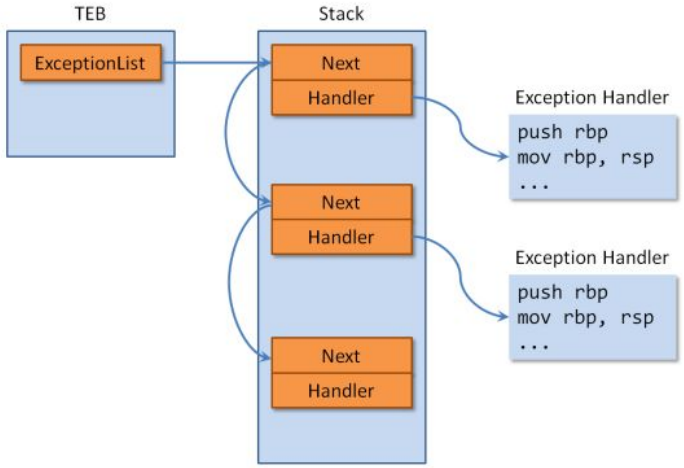
Использование стека процессором при обработке прерываний

Windows поддерживает специальную структуру, TEB (Thread Environment Block), где хранятся локальные данные потока. Эта структура доступна из Ring 3 через сегмент FS (или GS для x64). В самом начале этой структуры хранится указатель на список вложенных блоков “__try” и соответствующих им блоков “__except” и “__finally”. Компилятор генерирует код, добавляющий элемент в этот список при входе в блок “__try” и удаляющий при выходе из блока. (Примечание: справедливо только для x86. 64-х битный код “раскручивает” стек пользуясь сгенерированным компилятором описанием кода.)



Использование стека процессором при обработке прерываний





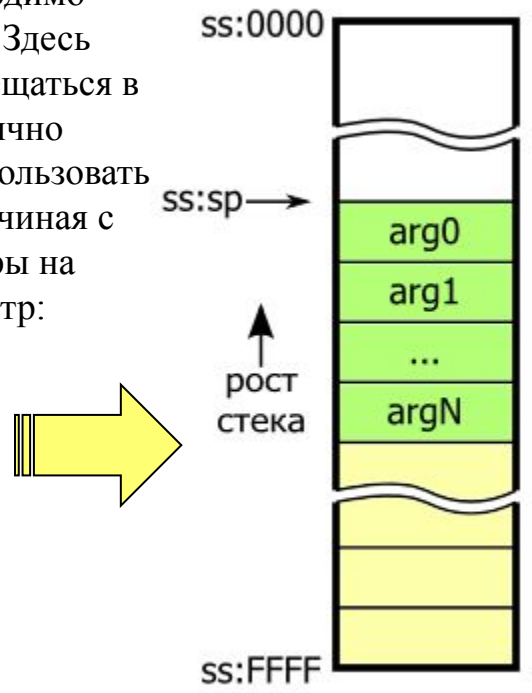
Получив управление, диспетчер исключений пользовательского режима по очереди опрашивает обработчики из списка, позволяя каждому из них обработать данное исключение. Найдя обработчик, согласившийся обработать исключение, диспетчер исключений выполняет «раскрутку» стека (Stack Unwinding) и передает управление выбранному обработчику. Этот механизм довольно детально описывался во всевозможных статьях о том, как работает SEH (Structured Exception Handling) (например здесь или здесь), так что я не буду останавливаться на этом детально. «Раскрутчик» стека отслеживает текущее состояние (указатель стека и команд, состояние регистров) в локальной копии структуры CONTEXT. По окончанию обработки исключения, состояние из структуры CONTEXT загружается в процессор, завершая обработку исключения.

Использование стека процессором при вызове

```
; Данные
arg0     dw 0
arg1     dw 12
argN     dw 345
;---------------------------------
---------------------------------
; Код
    push [argN]
    push ...
    push [arg1]
    push [arg0]
    call myproc
```

Перед вызовом процедуры параметры необходимо поместить в стек с помощью команды PUSH. Здесь существует два варианта: параметры могут помещаться в стек в прямом или в обратном порядке. Обычно используется обратный порядок и его я буду использовать в примерах. Параметры помещаются в стек, начиная с последнего, так что перед вызовом процедуры на вершине стека оказывается первый параметр:

Перед выполнением команды CALL стек будет иметь следующий вид:

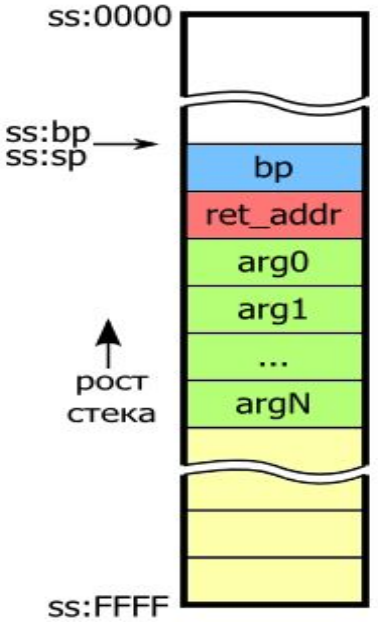


Использование стека процессором при вызове

Обращение к параметрам внутри процедуры

Для обращения к параметрам внутри процедуры обычно используют регистр BP. В самом начале процедуры содержимое регистра BP сохраняется в стеке и в него копируется значение регистра SP. Это позволяет «запомнить» положение вершины стека и адресовать параметры относительно регистра BP.

```
;Процедура
myproc:
  push bp
  mov bp,sp
  …
```



При выполнении кода процедуры стек будет иметь следующую структуру:

Здесь ret_addr обозначает адрес возврата, помещаемый в стек командой вызова процедуры, а bp – сохранённое значение регистра BP. В нашем случае стек имеет ширину 16 бит, поэтому первый параметр будет доступен как word[bp+4], второй как word[bp+6] и так далее.

```
mov ax,[bp+4]      ;AX = arg0
mov bx,[bp+6]      ;BX = arg1
add ax,[bp+8]      ;AX = AX + arg2
```

Не забудьте перед возвратом из процедуры восстановить значение BP из стека.

Использование стека процессором при возврате

Извлечение параметров из стека

```
push [arg1]
push [arg0]
call myproc
...
;-----------------------------------------------
;Процедура с двумя параметрами
myproc:
push bp
mov bp,sp
...
pop bp
ret 4        ;Из стека дополнительно извлекается 4 байта
```

После того, как процедура выполнилась, необходимо очистить стек, вытолкнув из него параметры. Тут тоже существует 2 способа: стек может быть очищен самой процедурой или кодом, который эту процедуру вызывал. Для первого способа используется команда RET с одним операндом, который должен быть равен количеству байтов, выталкиваемых из стека. В нашем случае он должен быть равен количеству параметров, умноженному на 2.

Для второго способа нужно использовать команду RET без операндов. Стек восстанавливается после выполнения процедуры путём прибавления значения к SP. С помощью такого способа программируются процедуры с переменным количеством параметров. Процедура не знает, сколько ей будет передано параметров, поэтому очистка стека должна выполняться вызывающим кодом.

```
push [arg1]
push [arg0]
call myproc2
add sp,4          ;Восстановление указателя стека
...
;-----------------------------------------------
;Процедура с двумя параметрами (не очищает стек)
myproc2:
push bp
mov bp,sp
...
pop bp
ret
```

Стековая машина. Стековая адресация.

```
        use16                   ;Генерировать 16-битный код
    org 100h                    ;Программа начинается с адреса 100h
            jmp start           ;Переход на метку start
    ;-------------------------------------------------------------------
                                ; Данные
                               a     dw 81
                               b     dw 273
                               x     dw ?
    ;-------------------------------------------------------------------
                                start:
                            push 3              ;c=3
                             push [b]           ;b
                             push [a]           ;a
                        call primer             ;Вызов процедуры
                           mov [x],ax           ;x=(a+b)/c

                            mov ax,4C00h        ;\
                    int 21h             ;/ Завершение программы

    ;-------------------------------------------------------------------
                ;Процедура с тремя параметрами: a, b, c.
    ;Вычисляет значение выражения (a+b)/c. Результат возвращается в AX.
                                primer:
                    push bp             ;Сохранение регистра BP
                            mov bp,sp           ;BP=SP
                                push dx

                             mov ax,[bp+4]      ;AX=a
                            add ax,[bp+6]       ;AX=(a+b)
                           cwd                  ;DX:AX=(a+b)
                          idiv word[bp+8]       ;AX=(a+b)/c

                                pop dx
                pop bp              ;Восстановление регистра BP
        ret 6               ;Возврат с извлечением параметров из стека
```

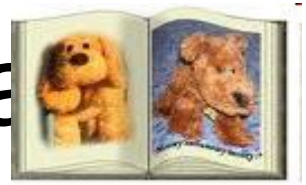
Соглашения вызова

Совокупность таких особенностей, как способ и порядок передачи параметров, механизм очистки стека, сохранение определённых регистров в процедуре и некоторых других называется соглашениями вызова. Соблюдение этих соглашений является важным, если вы из своей программы обращаетесь к компонентам, написанным на других языках программирования, вызываете функции ОС, или хотите из других языков вызывать процедуры, написанные на ассемблере.

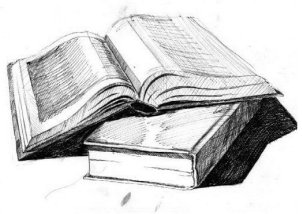
В качестве примера рассмотрим процедуру с тремя параметрами: a, b и c. Процедура вычисляет значение выражения (a+b)/c. Параметры передаются через стек в обратном порядке, результат возвращается в регистре AX, стек восстанавливается вызываемой процедурой. Все числа – 16-битные целые со знаком.

Главное – не запутаться со смещениями относительно BP.

Используемая литература



- Книга «Ассемблер. Учебник для ВУЗов», авторы Михаил Гук, Виктор Юров
- Книга «Архитектура ЭВМ»,автор Мюллер
- Книга «Процессоры *Pentium4, Athlon* и *Duron*», авторы Михаил Гук, Виктор Юров
- Книга «Архитектура ЭВМ», автор Танненбаум



- http://ru.wikipedia.org/wiki/Стек
- http://ru.wikipedia.org/wiki/Стековая_машина
- http://ru.wikipedia.org/wiki/Стек_вызовов
- http://blogs.technet.com/b/not-a-kernel-guy/archive/2008/10/16/3137048.aspx
- http://www.intuit.ru/department/security/mssec/23/
- http://www.intuit.ru/department/se/parallprog/8/4.html
- http://asmworld.ru/uchebnyj-kurs/025-peredacha-parametrov-cherez-stek/