Вопросы

1. Какие вы знаете соглашения о вызове?

Соглашение о вызовах определяет протокол взаимодействия вызывающей и вызываемой программ.

Тридцатидвухбитные соглашения о вызовах

Таблица 6.1

Соглашение	Параметры в регистрах	Порядок	Очистка стека	Изменяемые регистры	Неизменяемые регистры
cdecl		C	вызывающая программа	eax, ecx,edx, $st(0)-st(7)$, $x/y/zmm$	ebx, ebp, esi, edi
pascal		Pascal	функция		
winapi (stdcall)		C	функция		
gnu		C	this — функция, остальные — вызывающая программа		
gnu fastcall	ecx, edx	C	функция		
gnu regparm (3)	eax, edx, ecx	C	функция		
Borland fastcall	ecx, edx	Pascal	функция		
Microsoft fastcall	ecx, edx	C	функция		

Шестидесятичетырёхбитные соглашения о вызовах

Таблица 6.2

Соглашение	Параметры в регистрах	Порядок	Очистка стека	Изменяемые регистры	Неизменяемые регистры
GNU/Linux, BSD, Mac OS X, компиляторы GCC, Intel	rdi, rsi, rdx, rcx, $r8$, $r9$, $zmm0-$ $-zmm7$	C	вызывающая программа	rax, rcx, rdx, rsi, rdi, $r8-r11$, $st(0)-st(7)$, $x/y/zmm$	rbx, rbp, $r12-r15$
Microsoft Windows, компиляторы MinGW, Microsoft, Intel	$rcx/zmm0$, $rdx/zmm1$, $r8$ $/zmm2$, $r9$ $/zmm3$	C	вызывающая программа	rax, rcx, rdx, $r8-r11$, $st(0)-st(7)$, $x/y/zmm$, кроме младших частей 6–15	rbx, rbp, rsi, rdi, $r12-r15$, $xmm6-$ $-xmm15$

2. Какая команда передаёт управление подпрограмме?

Команда call ___

3. Какая команда возвращает управление вызывающей программе?

Команда ret

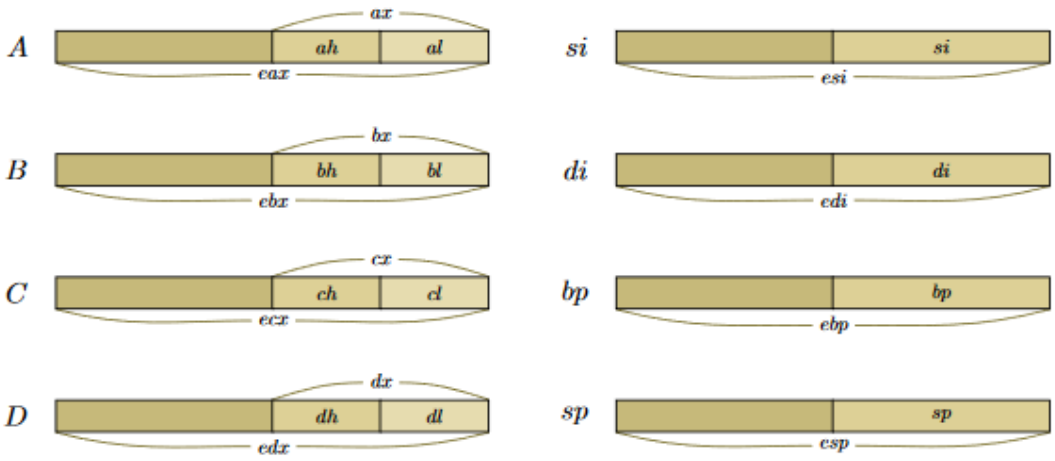
4. Что такое адрес возврата?

Команда call помещает адрес возврата в стек и переходит по адресу src.

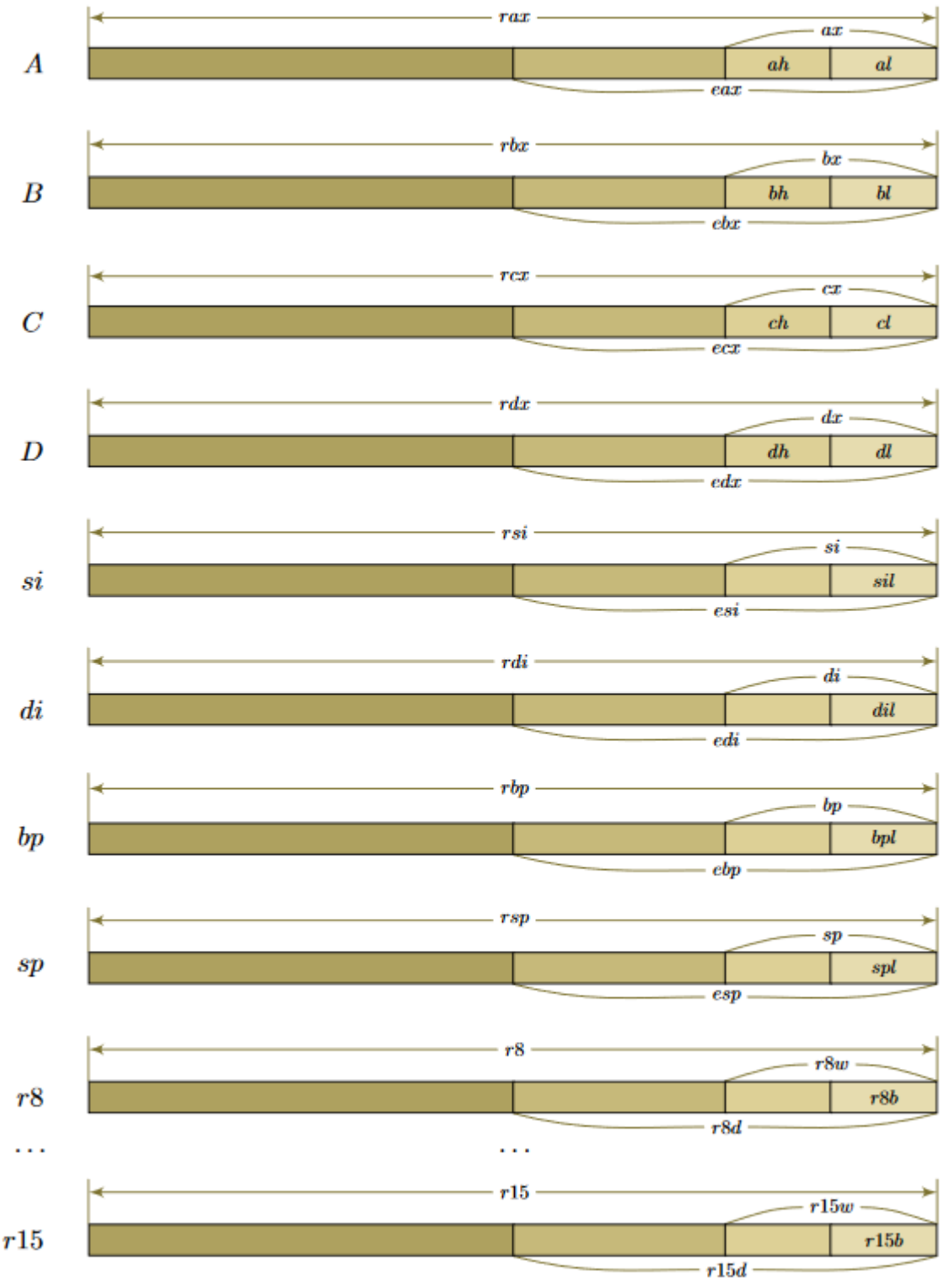
Команда ret снимает со стека адрес возврата и помещает его в указатель команд

5. Какие вы знаете регистры общего назначения?

Для 32-ого битного режима используются



Для 64-х битного режима



6. Какие вы знаете команды ассемблера x86?

Основные общие команды

Таблица 5.3

Команда	Действие
`nop` `nop srm`	Ничего не делает *(no operation)*
`mov src, dest`	Присваивание $dest = src$ *(move)*
`movabs imm64, dreg64`	В 64-битном режиме присваивание абсолютного 64-битного адреса $dreg64 = imm64$
`lea smem, dreg`	Вычисление адреса $smem$ и запись его в $dreg$ $dreg = \&smem$ *(load effective address)*
`xchg srm, dest`	Обмен значений srm и $dest$
Работа со стеком	
`push src`	Помещение src в стек (уменьшает указатель стека)
`pop dest`	Выталкивание значение из стека в $dest$ (увеличивает указатель стека)

В систему команд x86 входят три пары команд вызова/возврата.

1. Команды `call/ret` предназначены для вызова функций и процедур, описанных в самой программе и прикладных библиотеках.
2. Команды `int/iret` предназначены для программного обращения к прерыванию.
   В современной прикладной программе явный вызов программного прерывания обычно используется только для обращения к ядру операционной системы (системного вызова, подробнее в разделе 6.2.9).
3. Команды, предназначенные специально для системных вызовов.
   В тридцатидвухбитном режиме это `sysenter/sysexit`, в шестидесятичетырёхбитном — `syscall/sysret`.

Команды целочисленной арифметики

Таблица 5.5

Команда	Действие
`inc dest`	Инкремент $++dest$ $(dest = dest + 1)$
`dec dest`	Декремент $--dest$ $(dest = dest - 1)$
Сложение и вычитание	
`add src, dest`	Сложение $dest \mathrel{+}= src$ $(dest = dest + src)$
`adc src, dest`	Сложение с переносом из предыдущей части $dest \mathrel{+}= src + CF$ $\left(dest = dest + (src + CF)\right)$
`sub src, dest`	Вычитание $dest \mathrel{-}= src$ $(dest = dest - src)$
`cmp src, dest`	Вычитание $dest - src$ без изменения $dest$ (сравнение $dest$ и src)
`sbb src, dest`	Вычитание с переносом из предыдущей части $dest \mathrel{-}= src + CF$ $\left(dest = dest - (src + CF)\right)$
`neg dest`	Изменение знака $dest = -dest$
Расчёт линейной комбинации	
`lea` δ`(r1,r2,`σ`), dreg`	$dreg = r1 + \sigma \cdot r2 + \delta$ (не изменяет флагов)
Умножение и деление	
`mul srm`	Беззнаковое умножение $D{:}A = A \cdot srm$ (таблица 5.6)
`imul srm`	Знаковое умножение $D{:}A = A \cdot srm$ (таблица 5.6)
`imul srm, dreg`	Умножение $dreg \mathrel{*}= srm$ $(dreg = dreg \cdot srm)$
`imul imm, srm, dreg`	Знаковое умножение $dreg = imm \cdot srm$
`div srm`	Беззнаковое деление с остатком (таблица 5.6) $\begin{cases} A = (D{:}A)/srm \\ D = (D{:}A)\%srm \end{cases}$
`idiv srm`	Знаковое деление с остатком (таблица 5.6) $\begin{cases} A = (D{:}A)/srm \\ D = (D{:}A)\%srm \end{cases}$
Масштабирование (битовый сдвиг)	
`shr times, dest`	Беззнаковое деление $dest \mathrel{/}= 2^{times}$
`sar times, dest`	Знаковое математическое деление $dest \mathrel{/}= 2^{times}$ (остаток предполагается неотрицательным)
`shl times, dest` `sal times, dest`	Умножение $dest \mathrel{*}= 2^{times}$

Shr/Shl – битовые сдвиги вправо и влево

7. Какие вы знаете флаги?

flags

0	CF	Carry Flag	Флаг переноса (беззнакового переполнения)	Состояние
1	1	—	Зарезервирован	
2	PF	Parity Flag	Флаг чётности	Состояние
3	0	—	Зарезервирован	
4	AF	Auxiliary Carry Flag	Флаг вспомогательного переноса	Состояние
5	0	—	Зарезервирован	
6	ZF	Zero Flag	Флаг нуля	Состояние
7	SF	Sign Flag	Флаг знака	Состояние
8	TF	Trap Flag	Флаг трассировки	Системный
9	IF	Interrupt Enable Flag	Флаг разрешения прерываний	Системный
10	DF	Direction Flag	Флаг направления	Управляющий
11	OF	Overflow Flag	Флаг знакового переполнения	Состояние
12–13	IOPL	I/O Privilege Level	Уровень приоритета ввода-вывода	Системный
14	NT	Nested Task	Флаг вложенности задач	Системный
15	0	—	Зарезервирован	

eflags

16	RF	Resume Flag	Флаг возобновления	Системный
17	VM	Virtual-8086 Mode	Режим виртуального процессора 8086	Системный
18	AC	Alignment Check	Проверка выравнивания	Системный
19	VIF	Virtual Interrupt Flag	Виртуальный флаг разрешения прерывания	Системный
20	VIP	Virtual Interrupt Pending	Ожидающее виртуальное прерывание	Системный
21	ID	ID Flag	Проверка на доступность инструкции CPUID	Системный
22–31		—	Зарезервированы	

Флаг переноса (Carry Flag = CF), также флаг беззнакового переполнения.

Флаг чётности (Parity Flag = PF). Устанавливается, если младший байт результата команды содержит чётное число единиц, иначе — сбрасывается. Флаг чётности использовался для подсчёта контрольных сумм.

Флаг вспомогательного переноса (Auxiliary Carry Flag = AF), также используется название «флаг коррекции» (Adjust Flag = AF). Устанавливается, если арифметическая операция производит перенос (заём) из младшей тетрады младшего байта, т. е. из бита 3 в старшую тетраду при сложении (вычитании).

Флаг нуля (Zero Flag = ZF). Устанавливается, если результат операции — нуль, иначе — сбрасывается.

Флаг знака (Sign Flag = SF). Всегда равен значению старшего битарезультата. Этот бит интерпретируется как знаковый в некоторых арифметическихоперациях (0/1 — число положительное/отрицательное).

Флаг знакового переполнения (Overflow Flag = OF). Устанавливается, если при знаковой интерпретации результат операции не помещается в операнд, иначе — сбрасывается.