КОНСТРУИРОВАНИЕ ПРОГРАММ

Лекция № 20

64-битный режим

+375 17 293 8039 (505a-5)

+375 17 320 7402 (ОИПИ НАНБ)

prep@lsi.bas-net.by

ftp://student:2ok*uK2@Rwox@lsi.bas-net.by/

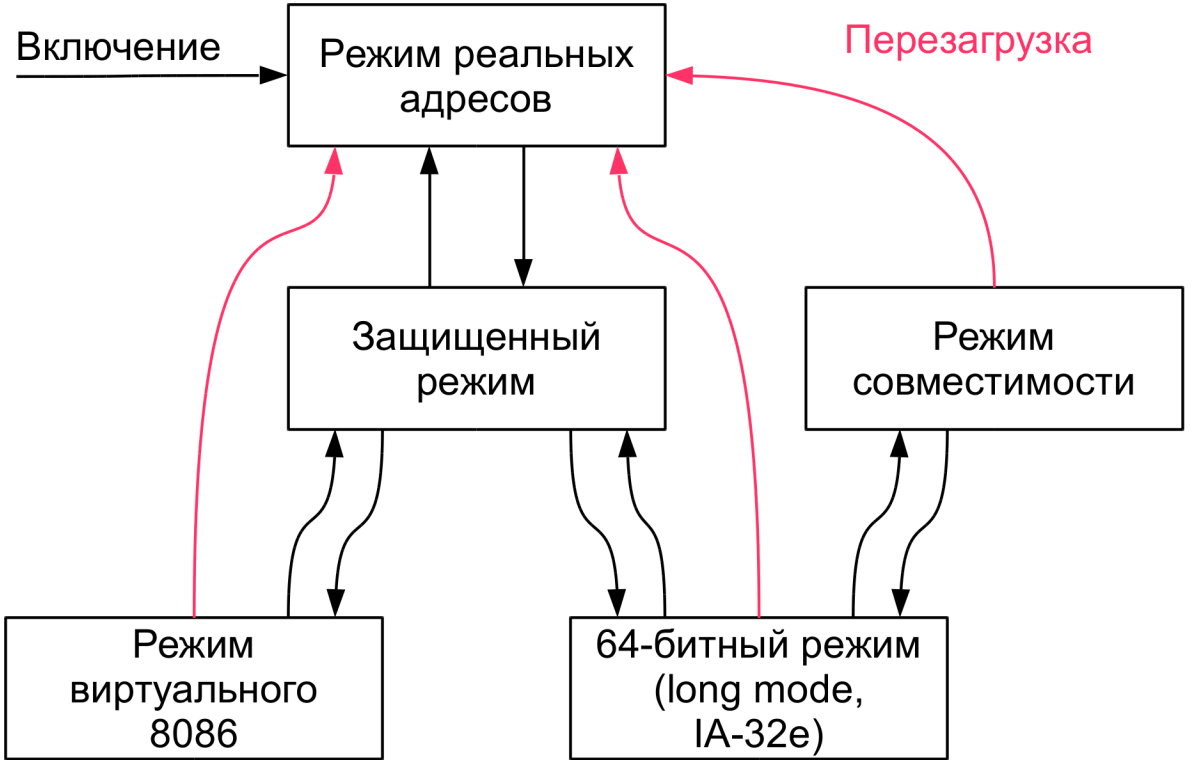
Кафедра ЭВМ, 2022

2022.05.13

Оглавление

Режимы процессора x86_64

Режим процессора x86_64	Intel	AMD	Разрядность, бит		Объем
			Адрес	Данные	
реальный			20	16	1М+64к
32 (защищённый)	IA-32	legacy mode	32(36)	32	4(64)Г
виртуальный 8086			16	16	
x64	IA-32e	long mode	64	32(64)	4096(64)Т
совместимости с 32			32(16)	32(16)	
SMM					



Технология AMD64 - это 64-битная архитектура микропроцессора и соответствующий набор инструкций, разработанный компанией AMD.

AMD64 - это расширение архитектуры x86 под 64-разрядную архитектуру, с полной обратной совместимостью.

Данная архитектура и набор инструкций были лицензированы (с незначительными изменениями) основному конкуренту AMD — компании Intel под названием **EM64T**.

Такой шаг компании Intel был более чем правильным, т. к. две различные 64-разрядные архитектуры привели бы только к замедлению прогресса в этой области, в частности при разработке 64-разрядных операционных систем.

Согласно документации Intel 64-битный режим процессора официально называется IA-32e, а согласно документации AMD - long mode.

Далее термином «long mode» обозначается сам 64-битный режим как таковой, со всеми его подрежимами.

Под словами «64-битный режим» следует понимать именно тот подрежим, в котором выполняется 64-разрядный код.

Общий обзор

Режим long mode имеет два подрежима:
- 64-битный режим;
- режим совместимости.

Объект	Размер по умолчанию, байт	
	64-битный	совместимости
Адрес	8	4
Операнд	4	4

Режим совместимости (32-битный подрежим режима long mode/IA-32e)

Режим совместимости включается путём установки специального бита в дескрипторе сегмента кода.

Режим совместимости позволяет запускать большинство устаревших 16-битных и 32-битных приложений без повторной компиляции в 64-битной операционной системе.

Среда выполнения режима совместимости такая же, как и для защищенного режима

Режим совместимости также поддерживает все уровни привилегий, которые поддерживаются в 64-битном и защищенном режимах.

Приложения, которые работают в режиме Virtual 8086 или используют аппаратное управление задачами, не будут работать в этом режиме.

Режим совместимости включается операционной системой (ОС) на основе характеристик сегмента кода. Это означает, что одна 64-битная ОС может поддерживать 64-битные приложения, работающие в 64-битном режиме, и устаревшие 32-битные приложения (не перекомпилированные для 64-битных), работающие в режиме совместимости.

Режим совместимости аналогичен 32-битному защищенному режиму.

Приложения обращаются только к первым 4 ГБ линейного адресного пространства.

В режиме совместимости используются 16-битные и 32-битные адреса и размеры операндов. Подобно защищенному режиму, этот режим позволяет приложениям получать доступ к физической памяти размером более 4 ГБ с использованием PAE (Physical Address Extensions).

64-битный режим (64-битный подрежим режима long-mode IA-32e)

64-битный режим режим позволяет 64-битной операционной системе запускать приложения, написанные для доступа к 64-битному линейному адресному пространству.

В 64-битном режиме количество регистров общего назначения и регистров расширения SIMD увеличивается с 8 до 16.

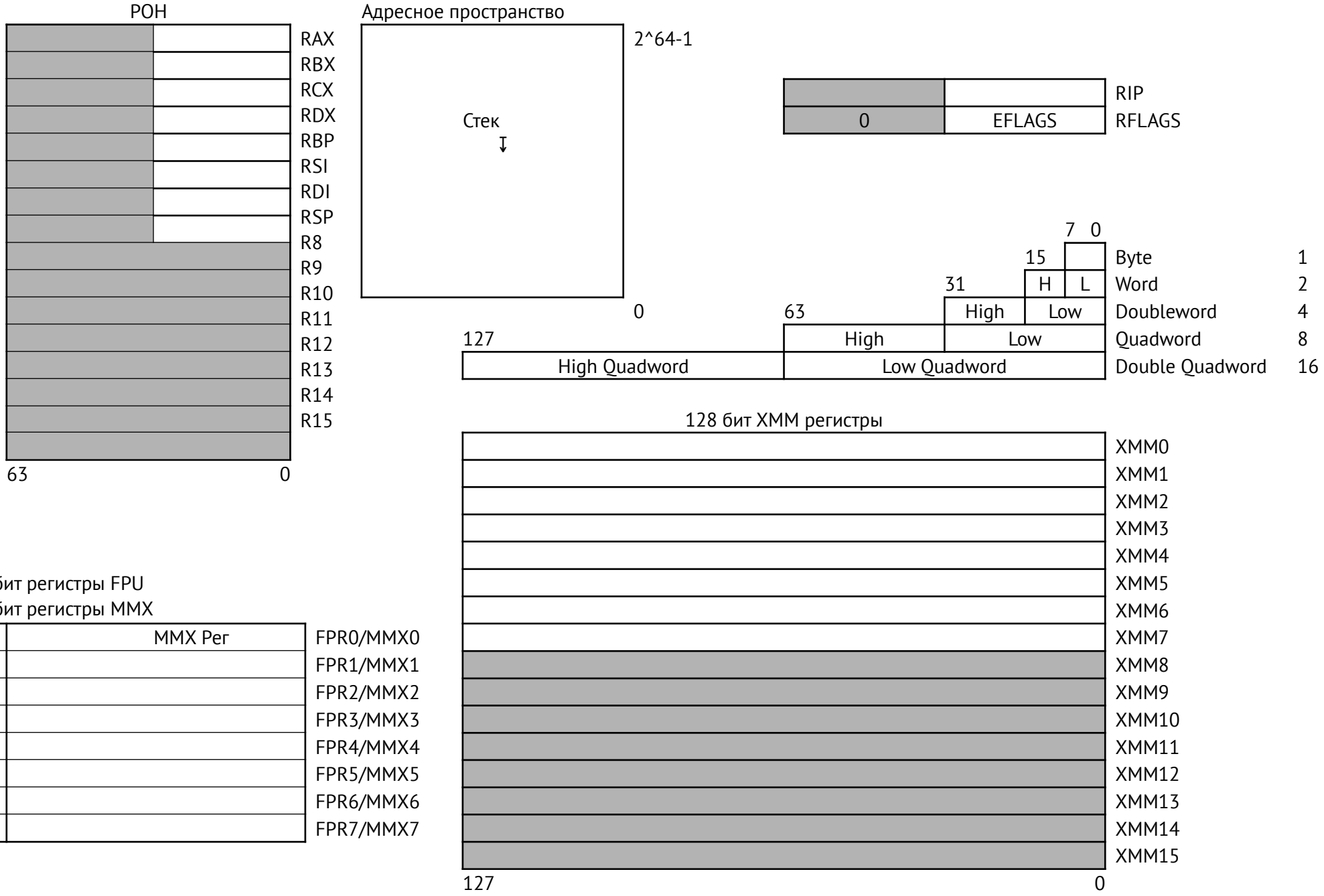
Регистры общего назначения расширены до 64 бит. В этом режиме также вводится новый префикс кода операции (REX) для доступа к расширениям регистров.

64-битный режим включается операционной системой на основе характеристик сегментов кода.

В 64-битном режиме процессора доступны следующие регистры процессора:
Регистры общего назначения:

64 бит	**32 бит**	**16 бит**	**8 бит**
RAX	EAX	AX	AL/AH
RBX	EBX	BX	BL/BH
RCX	ECX	CX	CL/CH
RDX	EDX	DX	DL/DH
RSP	ESP	SP	SPL
RBP	EBP	BP	BPL
R8..R15	R8D..R15D	R8W..R15W	R8B..R15B (R8L..R15L)
RSI	ESI	SI	SIL
RDI	EDI	DI	DIL

RIP — 64-разрядный указатель инструкции;
CS, DS, SS, ES, FS, GS — 16 разрядные сегментные регистры (6);
RFLAGS — 64-разрядный регистр флагов;
ST0..ST7 — 80-битные регистры математического сопроцессора;
MM0-MM7 — 64-битные MMX-регистры;
XMM0-XMM15 — 128-разрядные XMM-регистры SSE;
MXCSR — 32-битный регистр контроля SSE;
CR0..CR4 и CR8 — 64-разрядные регистры управления;
GDTR, LDTR, IDTR — регистры-указатели системных таблиц;
TR — регистр задачи;
DR0..DR3, DR6, DR7 — 64-разрядные регистры отладки;
MSR-регистры.


РОН
RAX
RBX
RCX
RDX
RBP
RSI
RDI
RSP
R8
R9
R10
R11
R12
R13
R14
R15
63
0
Адресное пространство
2^64-1
Стек
0
RIP
0
EFLAGS
RFLAGS
7 0
15
31
63
127
Byte 1
Word 2
Doubleword 4
Quadword 8
Double Quadword 16
H
L
High
Low
High Quadword
Low Quadword
128 бит XMM регистры
XMM0
XMM1
XMM2
XMM3
XMM4
XMM5
XMM6
XMM7
XMM8
XMM9
XMM10
XMM11
XMM12
XMM13
XMM14
XMM15
127
0
80 бит регистры FPU
64 бит регистры MMX
MMX Per
FPR0/MMX0
FPR1/MMX1
FPR2/MMX2
FPR3/MMX3
FPR4/MMX4
FPR5/MMX5
FPR6/MMX6
FPR7/MMX7

Доступ к 64-битным регистрам и новым регистрам осуществляется через специальный **REX**-префикс. Таким образом, все опкоды команд, которые работают с 64-битными регистрами, увеличиваются в размере как минимум на 1 байт, и возникает серьёзная проблема оптимизации кода, поэтому

Рекомендуется везде, где возможно, использовать 32-битные регистры

Размер адреса по умолчанию составляет 64 бита, а размер операнда по умолчанию — 32 бита. Размер операнда по умолчанию можно переопределить для каждой инструкции с помощью префикса кода операции **REX** в сочетании с префиксом переопределения размера операнда.

Префиксы **REX** позволяют указать 64-битный операнд при работе в 64-битном режиме.

Благодаря использованию этого механизма многие существующие инструкции были доработаны, чтобы разрешить использование 64-битных регистров и 64-битных адресов. Тем не менее, система команд в целом почти не претерпела сильных изменений, в частности, формат команд остался тем же.

Команды **PUSHAD**/**POPAD**, сохраняющая и восстанавливающая содержимое всех 8 регистров общего назначения (EAX, EBX, ECX, EDX, ESP, EBP, ESI, EDI), не поддерживаются;

Команды **PUSH** и **POP** могут работать только с 64- и 16-битными операндами;

При помещении в стек 16-битный операнд не расширяется до 64 бит, но при этом указатель на вершину стека перестаёт быть выровненным на 8 байт.

Регистр **EFLAGS** расширен до 64 бит и теперь называется **RFLAGS**. Однако состав его полей остался без изменений.

Из-за введения 64-битной системы команд теперь стало проще работать с 64-битными числами, и некоторые операции можно произвести одной командой.

Указатель инструкции в 64-битном режиме

Указателем инструкции в 64-битном режиме становится регистр **RIP**.

Этот регистр содержит 64-битное смещение следующей инструкции, которая должна быть выполнена следующей.

64-битный режим также поддерживает метод, называемый относительной адресацией RIP. Используя этот метод, эффективный адрес определяется путем добавления смещения к RIP следующей инструкции.

Регистр флагов RFLAGS

0	CF	Флаг переноса
1		Зарезервирован (всегда установлен)
2	PF	Флаг чётности
3		Зарезервирован (всегда сброшен)
4	AF	Дополнительный флаг переноса
5		Зарезервирован (всегда сброшен)
6	ZF	Флаг нулевого результата
7	SF	Флаг знака результата
8	TF	Флаг трассировки
9	IF	Флаг разрешения прерывания
10	DF	Флаг направления
11	OF	Флаг переполнения
12,13	IOPL	Текущий уровень привилегий ввода/вывода
14	NT	Флаг вложенности задачи
15		Зарезервирован (всегда сброшен)
16	RF	Флаг возобновления
17	VM	Флаг виртуального процессора 8086
18	AC	Флаг проверки выравнивания адреса
19	VIF	Флаг виртуального прерывания
20	VIP	Флаг отложенного виртуального прерывания
21	ID	Флаг разрешения идентификации процессора
22..31(63)		Зарезервированы (всегда сброшены)

Организация памяти в режиме 64-бит

Архитектура Intel 64 поддерживает физическое адресное пространство размером более 64 ГБ, однако, фактический размер физического адреса процессоров IA-32 зависит от реализации.

В 64-битном режиме есть архитектурная поддержка 64-битного линейного адресного пространства. Однако процессоры, поддерживающие архитектуру Intel 64, могут иметь менее, чем полную 64-разрядную версию (см. Раздел 3.3.7.1).

Линейное адресное пространство отображается в физическое адресное пространство процессора через страничный механизм PAE.

В 64-битном режиме адрес считается каноническим, если адресные биты от 63 до наиболее значимого бита, реализованного микроархитектурой, установлены либо на все единицы, либо на все нули.

Архитектура Intel 64 определяет также 64-битный линейный адрес, однако реализации могут поддерживать меньше.

Первая реализация процессоров IA-32 с архитектурой Intel 64 поддерживает 48-битный линейный адрес. Это означает, что канонический адрес должен иметь биты с 63 по 48, установленные в нули или единицы (в зависимости от того, является ли бит 47 нулем или единицей).

Хотя некотороые реализации могут не использовать все 64 бита линейного адреса, они все равно проверяют биты с 63 до наиболее значимого реализованного бита, чтобы увидеть, находится ли адрес в канонической форме. Если ссылка на линейную память не в канонической форме, будет сгенерировано исключение.

В большинстве случаев генерируется исключение общей защиты (#GP).

В случае явных или подразумеваемых ссылок на стек генерируется ошибка стека (#SS).

Инструкции, которые подразумевают ссылки на стек, по умолчанию используют регистр сегмента SS. К ним относятся инструкции, относящиеся к PUSH/POP, и инструкции, использующие RSP/RBP в качестве базовых регистров. В этих случаях каноническая ошибка — это #SS.

Если инструкция использует базовые регистры RSP/RBP и для указания сегмента, отличного от SS, использует префикс переопределения сегмента, каноническая ошибка генерирует #GP (вместо #SS).

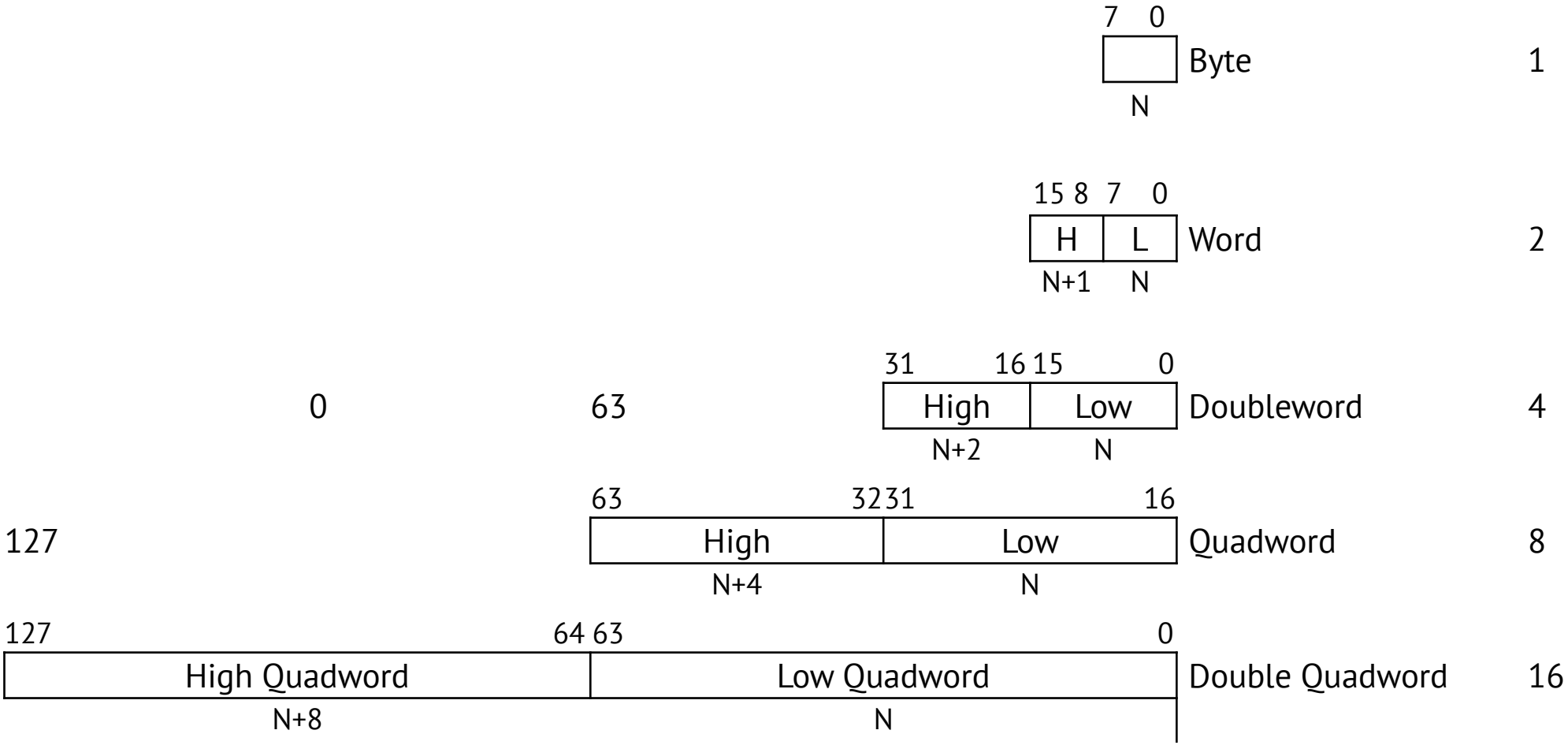
В 64-битном режиме в этой ситуации применимы только переопределения сегментов FS и GS.

Другие префиксы отмены сегмента (CS, DS, ES и SS) игнорируются. Это также означает, что игнорируется переопределение сегмента SS, примененное к ссылке на не стековый регистр. Такая последовательность по-прежнему создает #GP для канонической ошибки (а не #SS).

```
0000007F 488B4510                       mov  rax, [rbp + 16]
00000083 64488B5D10                     mov  rbx, [FS: rbp + 16]
00000088 65488B4D10                     mov  rcx, [GS: rbp + 16]
0000008D 26488B5D10                     mov  rbx, [ES: rbp + 16]
         ******************  warning: es segment base generated, but will be ignored
in 64-bit mode
0000008D 26488B5D10                     db   0x26, 0x48, 0x8B, 0x5D, 0x10
```

Если таки обеспечить генерацию прямым указанием кода, то при запуске получим

```
$ ./hello-GP
Ошибка сегментирования (стек памяти сброшен на диск)
```



Тип данных **quadword** (квадрослово) было введено в архитектуру IA-32 в процессоре Intel486.

Тип данных **double quadword** (двойное квадрослово) было введено в процессоре Pentium III с расширениями SSE.

Выравнивание слов, двойных слов, квадрослов и двойных квадрослов

Естественными границами для слов, двойных слов и четверных слов являются адреса с четными номерами, адреса, делимые на четыре, и адреса, делимые на восемь, соответственно.

Слова, двойные слова и квадрослова не нужно выравнивать в памяти по естественным границам, однако для повышения производительности программ структуры данных (особенно стеки) должны быть по возможности выровнены по естественным границам. Причина этого в том, что процессору требуется два доступа к памяти для выполнения невыровненного доступа к памяти, в то время как выровненный доступ требует только одного доступа к памяти.

Операнд слова или двойного слова, пересекающий 4-байтовую границу, или операнд четверного слова, пересекающий 8-байтовую границу, считается невыровненным и требует для доступа двух отдельных циклов шины памяти.

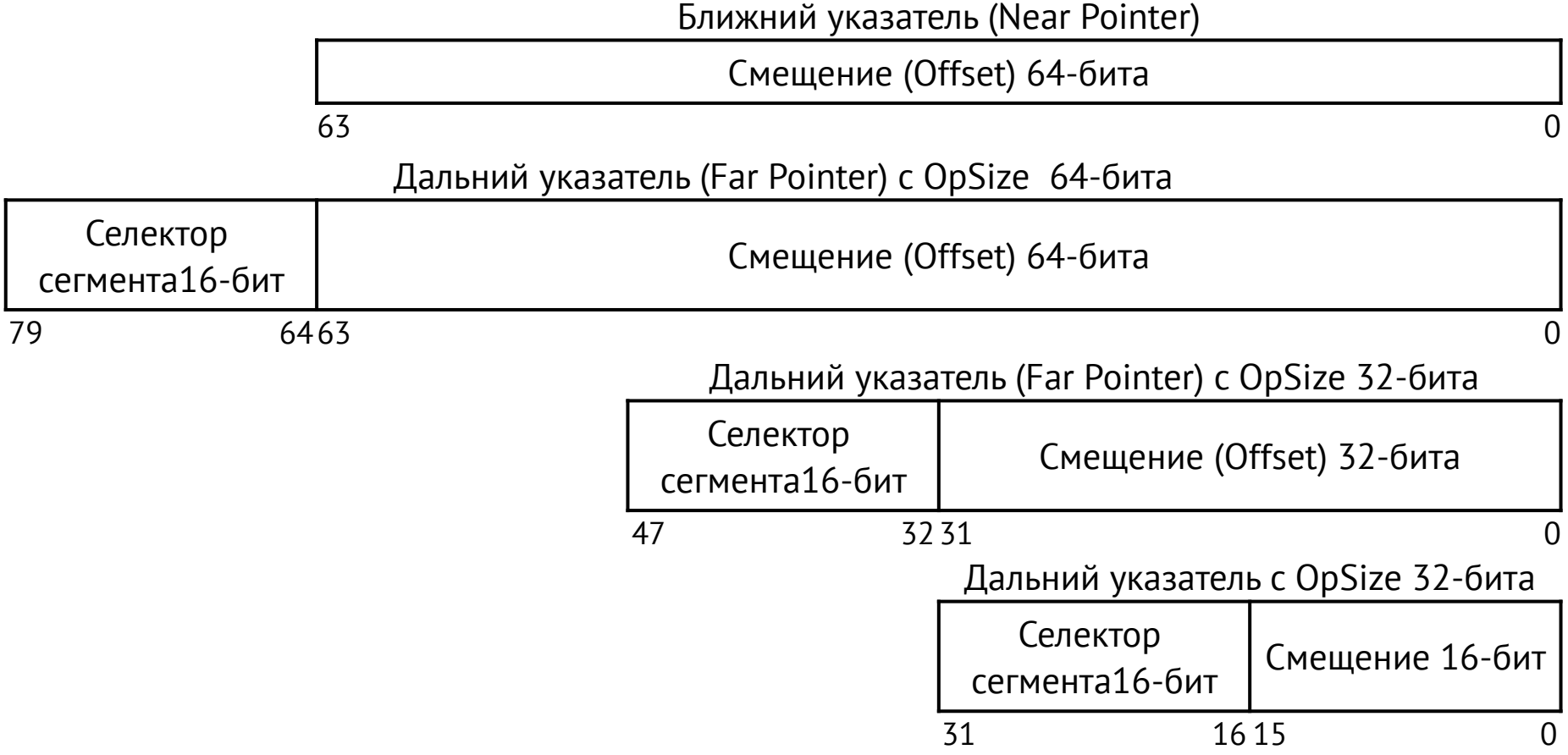
Некоторые инструкции, которые работают с двойными квадрословами, требуют, чтобы операнды памяти были выровнены по естественной границе. Эти инструкции генерируют исключение общей защиты (#GP), если указан невыровненный операнд.

Естественной границей для двойного четверного слова является любой адрес, без остатка делимый на 16. Другие инструкции, которые работают с двойными квадрословами, разрешают невыровненный доступ (без генерации исключения общей защиты). Однако для доступа к невыровненным данным из памяти требуются дополнительные циклы шины памяти.

Типы данных указателя в 64-битном режиме

В 64-битном режиме ближний (near) указатель составляет 64 бита. Это эквивалентно эффективному адресу. Дальние (far) указатели в 64-битном режиме могут иметь одну из трех форм:

- 16-битный селектор сегмента, 16-битное смещение, если размер операнда 32 бита
- 16-битный селектор сегмента, 32-битное смещение, если размер операнда 32 бита
- 16-битный селектор сегмента, 64-битное смещение, если размер операнда 64 бит



Поведение стека в 64-битном режиме

В 64-битном режиме вычисление адреса, который ссылается на сегмент SS, обрабатывается так, как если бы база сегмента была равна нулю. Поля (базовые, предельные и атрибуты) в регистрах дескрипторов сегмента игнорируются. SS DPL модифицируется так, что он всегда равен CPL.

Регистры E(SP), E(IP) и E(BP) преобразованы в 64-битные и переименованы в RSP, RIP и RBP соответственно.

Некоторые формы инструкций загрузки сегмента недопустимы (например, LDS, POP ES).

Инструкции PUSH/POP увеличивают/уменьшают стек с шагом 64-бита (квадрослово).

Когда в 64-битный стек помещается содержимое сегментного регистра, указатель автоматически выравнивается по 64 битам (аналогично тому, как это имеет место в случае стека с 32-битного размера).

Функции ветвления в 64-битном режиме

64-битные режим расширил и адаптировал механизмы переходов под в 64-битное линейное адресное пространство.

- семантика ближних переходов в 64-битном режиме переопределена;
- в 64-битном режиме и режиме совместимости доступны 64-битные дескрипторы шлюза для дальних вызовов;

В 64-битном режиме размер операнда для всех ближних переходов (CALL, RET, JCC, JCXZ, JMP и LOOP) принудительно устанавливается равным 64 битам.

Эти инструкции обновляют 64-битный RIP без необходимости в префиксе размера операнда префикса REX.

Эффективный размер операнда контролирует следующие аспекты ближних переходов:

-усечение размера указателя инструкций;

- шаг pop/push в процессе выполнения CALL или RET;
- шаг инкремента/декремента указателя стека в процессе выполнения CALL или RET;
- размер операнда инструкции косвенного перехода.

В 64-битном режиме все вышеперечисленные действия принудительно устанавливаются на 64 бита независимо от префиксов размера операнда (префиксы размера операнда автоматически игнорируются).

Однако поле смещения для относительных ветвей по-прежнему ограничено 32 битами, а размер адреса для ближайших ветвей в 64-битном режиме принудительно не устанавливается.

Размеры адресов влияют на размер RCX, используемого для JCXZ и LOOP.

Они также влияют на вычисление адресов для косвенных переходов в памяти.

По умолчанию такие адреса 64-битные, но они могут быть заменены на 32 бита с помощью префикса размера адреса.

Программное обеспечение обычно использует дальние переходы для изменения уровней привилегий.

Унаследованная архитектура IA-32 предоставляет механизм шлюза вызова, позволяющий программному обеспечению переходить с одного уровня привилегий на другой, хотя шлюзы вызова также могут использоваться для переходов, которые не изменяют уровни привилегий.

Когда используются шлюзы вызова, селекторная часть прямого или косвенного указателя ссылается на дескриптор шлюза, а смещение в инструкции игнорируется. Смещение в сегменте кода адреса назначения берется из дескриптора шлюза вызова.

64-битный режим переопределяет значение типа 32-битного дескриптора шлюза вызова на 64-битный дескриптор шлюза вызова и увеличивает размер 64-битного дескриптора для хранения 64-битного смещения. Дескриптор шлюза в 64-битном режиме допускает дальние переходы, которые ссылаются на любое место в поддерживаемом линейном адресном пространстве. Эти шлюзы вызова также содержат селектор целевого кода (CS), позволяя изменять уровень привилегий и размер по умолчанию в результате перехода шлюза.

Поскольку непосредственные операнды в инструкциях перехода обычно имеют размер 32 бита, единственный способ указать полный 64-битный абсолютный RIP в 64-битном режиме — это косвенный переход. По этой причине в 64-битном режиме прямые дальние переходы исключены из набора команд.

Таким образом в системе команд 64-битного режима усложнилось использование команд дальней передачи управления — команд типа **JMP selector16:offset64** не существует

Для межсегментной передачи управления (она используется для перезагрузки регистра CS) используется способ, приведённый ниже

```
target:
dq    offset
dw    selector
. . .
jmp tbyte[target]
```

```
db - 1 байт
dw - 2 байта - 1 слово
dd - 4 байта - 2 слова
dq - 8 байт  - 4 слова
dt - 10 байт
```

Такой способ был доступен и в защищённом режиме, но в 64-битном режиме он является единственно возможным.

Непосредственное 64-битное значение можно заносить только в регистр — команды формата

```
mov [address], value64  ;
```

в отличие от защищенного режима

```
mov [address], value32  ;
```

не поддерживаются.

Если необходимо занести 64-битное значение в ячейку памяти, следует сначала занести его в регистр, а потом из регистра в память. Таким образом, ряд действий, которые в защищённом режиме выполнялись за один шаг, теперь будут выполняться в два шага.

В 64-битном режиме введён режим адресации относительно **RIP**.

В случае адресации относительно **RIP** используются не 64-битные адреса данных и переходов, а 32-битные. Это позволяет сократить размер кода до 40%.

Значительные изменения в **long mode** претерпела сегментация — она поддерживается только частично.

Все сегменты начинаются с адреса 0h и покрывают всё 64-битное адресное пространство, проверка лимита не производится.

В дескрипторах данных игнорируются почти все поля.

В дескрипторе кода игнорируется большинство полей.

Механизм многозадачности аппаратно не поддерживается и он реализуется программно.

Небольшие изменения претерпел механизм трансляции страниц, в большей степени из-за того, что увеличилось количество виртуальной памяти. Теперь фактически доступно до 2^{48} байт (256 Тб) виртуальной памяти и 2^{52} байт физической.

64-битный режим также расширяет семантику инструкций SYSENTER и SYSEXIT, таким образом, чтобы эти инструкции могли работать в 64-битном пространстве памяти.

В этом режиме также представлены две новые инструкции SYSCALL и SYSRET, которые действительны только в 64-битном режиме.

```
%ifidn __OUTPUT_FORMAT__, elf32
%define syscall int 80h                 ; 0F05
%elifidn __OUTPUT_FORMAT__, elf64
%define syscall syscall                 ; CD80
%else
%error Выходной формат __OUTPUT_FORMAT__ не поддерживается
%endif
```

Прерывания и исключения в 64-битном режиме

Чтобы обеспечить поддержку 64-разрядных операционных систем и приложений, 64-разрядные расширения расширяют унаследованный механизм обработки прерываний и исключений IA-32.

Изменения включают:

- все обработчики прерываний, на которые указывает IDT, представляют собой 64-битный код.

- размер элемента, помещаемого в стек прерываний фиксирован и составляет 64 бита. Процессор использует 8-байтовые области в памяти, расширенные нулем.

- в случае прерывания указатель стека (**SS:RSP**) помещается в стек безусловно. В унаследованных средах этот push является условным и зависит от изменения текущего уровня привилегий (CPL).

- если есть изменение в CPL, Новый SS устанавливается в NULL.

- изменяется поведение IRET.

- используется новый механизм переключения стека прерываний и новый механизм переключения стека теневого прерывания.

- выравнивание кадра стека прерываний другое.

Supermicro H11SSW-NT

1. Single AMD EPYC™ 7001/7002* Series Processor (*AMD EPYC 7002 series drop-in support requires board revision 2.x)
2. 4TB Registered ECC DDR4 3200MHz SDRAM in 16 DIMMs (Board revision 2.x required)
3. Expansion slots:
    - 1 PCI-E 3.0 x32 Left Riser Slot
    - 1 PCI-E 3.0 x16 Right Riser Slot
    - M.2 Interface: 2 PCI-E 3.0 x2
    - M.2 Form Factor: 2280, 22110
    - M.2 Key: M-key
4. 12 NVMe via SlimSAS, 4 NVMe or 16 SATA3 via SlimSAS, 2 M.2, 2 SATA3
5. 2x 10GBase-T LAN Ports via Broadcom BCM57416, 1 Realtek RTL8211E PHY (dedicated IPMI)
6. ASPEED AST2500 BMC graphics
7. Up to 7 USB 3.0 ports (4 rear + 2 front + 1 Type A)
8. 6 PWM 4-pin Fans with tachometer status monitoring

Сегментация в long mode

Сегментация в **long mode** фактически отключена.

Проверка лимита сегментов не осуществляется — все сегменты кода и данных имеют базовый адрес равный нулю.

В целях поддержки совместимости в **long mode** доступ к теневым частям сегментных регистров FS и GS можно получить через MSR-регистры

```
IA32_FS_BASE
IA32_GS_BASE
```

Таким образом, имеется возможность изменить базовый адрес для сегментов, описываемых в FS и GS, что ликвидирует возможные проблемы при переносе 32-битных систем на 64-битную платформу (в системах Win32 регистр FS указывает на блок информации о текущем потоке).

Механизм трансляции страниц

В **long mode** есть только один режим трансляции страниц — режим **PAE**. Он всегда должен быть включён.

Перейти в **long mode** при отключённом режиме **PAE** невозможно и отключить его тоже нельзя.

В 64-битном режиме доступны 2^{48} байт виртуальной памяти и 2^{52} байт физической памяти.
Страницы могут быть размером 4 Кб, 2 Мб и 1 Гб, причём допускается их комбинирование.
При трансляции адреса используются 4 типа системных таблиц.
1) таблица страниц
2) таблица каталогов страниц
3) таблица указателей на каталоги страниц
4) карта страниц четвёртого уровня (PML4).

Регистр CR3 расширен до 8 байт и может хранить 64-битный адрес PML4.

Принципы работы с виртуальной памятью в 64-битном режиме

64-битный режим может работать только с включённым механизмом трансляции страниц и почти вся защита данных и кода операционной системы от пользовательского кода почти полностью возлагается на уровень страниц.

Работа с виртуальной памятью стала немного сложнее, чем в защищённом режиме, поскольку увеличилось количество таблиц и структур для описания страниц памяти, но основные принципы работы с виртуальной памятью не изменились.

В 64-битном режиме доступно до 2^{52} байт физической памяти и 2^{48} байт виртуальной.

Максимальная разрядность физического адреса равная 52 поддерживается не всеми моделями процессора, поэтому для получения максимальной разрядности физического адреса, поддерживаемой процессором, на котором выполняется программа, следует использовать инструкцию **CPUID**.

Ограничение на разрядность физического адреса равную 52 является ограничением самой архитектуры процессора.

Новые процессоры не будут поддерживать более 2^{52} байт памяти.

Ограничение в 48 бит не является таким жёстким ограничением, как ограничение на разрядность физического адреса, и возможно, что в будущих моделях процессоров будут введены дополнительные структуры для трансляции виртуальных адресов более 48 бит.

В 64-битном режиме доступны следующие размеры страниц:
- 4 Кб (виртуальный адрес делится на пять частей);
- 2 Мб (виртуальный адрес делится на четыре части).
- 1 Гб (виртуальный адрес делится на три части)
Последний размер поддерживается не всеми процессорами.

В long mode флаг PSE[1] в регистре CR4 игнорируется и ни на что не влияет.
В **long mode** может быть использовано 2^{48} байт виртуальной памяти.

Все используемые виртуальные адреса должны быть в т.н. канонической форме — все старшие 16 бит адреса должны быть либо нулями, либо единицами. В любом другом случае мы получим исключение общей защиты (#GP) либо ошибку стека (#SS).

Данная форма адреса является мерой для обеспечения совместимости с будущими моделями процессоров. Связано это с тем, что большинство операционных систем делят адресное пространство на две части: в младшей части — память обычных процессов, в старшей — память ядра.

Таким образом, запрет использовать старшие 16 бит виртуального адреса обеспечивает совместимость с будущими моделями процессоров, в которых эффективная часть виртуального адреса будет большей размерности.

1 PSE (Page Size Extensions) разрешает использование страниц размером 4 Мб.
Данный флаг актуален только в обычном режиме трансляции адресов в защищённом режиме процессора.

Проверки защиты

Поскольку сегментная организация памяти в **long mode** фактически отсутствует, основные защитные механизмы в **long mode** возлагаются на механизм страничного преобразования.

Защиту на уровне страниц в **long mode** осуществляют следующие биты:

1. **R/W** в элементе каждой таблицы.
2. **U/S** в элементе каждой таблицы.
3. **NX** в элементе каждой таблицы.
4. **WP** в регистре CR0.
5. **NXE** в MSR регистре IA32 EFER.

Пользовательский код будет иметь доступ к странице только в том случае, когда бит **U/S** выставлен в каждой структуре, описывающей искомую страницу.

Привилегированный код будет иметь доступ к странице независимо от того, выставлен ли бит **U/S**.

Пользовательский код будет иметь возможность изменять данные только в том случае, когда бит **R/W** выставлен в каждой структуре, описывающей данную страницу.

Так же, как и в защищённом режиме, привилегированный код по умолчанию (если мы не изменяли бит **WP**[2] (16) в регистре CR0) может изменять даже те страницы, которые помечены «только для чтения».

Когда бит **WP** сброшен, привилегированный код (код на уровнях 0, 1 и 2) имеет доступ на изменение ко всем страницам в системе.

Если бит **WP** (16) в регистре **CR0** выставлен, то привилегированный код работает по тем же правилам, что и пользовательский код, т. е. может изменить данные на странице только в том случае, когда бит **R/W** выставлен в каждой структуре, описывающей искомую страницу.

NX — Если этот бит **NX** выставлен, то на странице, описываемой данной структурой, запрещено выполняться любому коду, а также запрещено передавать управление коду, который находится на этой странице.

Код сможет выполняться на странице, только когда бит **NX** сброшен в каждой структуре, описывающей искомую страницу. Действие флага **NX** не зависит от флага **WP** и одинаково влияет на пользовательский и привилегированный код.

Просто выставить бит **NX** в структурах, недостаточно для активирования данной возможности. Чтобы эта возможность заработала, следует установить в единицу бит **NXE** (одиннадцатый бит) в MSR-регистре **IA32_EFER**.

---

2) **WP** (Write Protect — бит 16) включает защиту от записи для кода, выполняющегося на нулевом уровне привилегий. Если этот бит выставлен, то код, выполняющийся на нулевом уровне привилегий, не сможет производить операцию записи на страницы, помеченные «только для чтения».

Если флаг WP сброшен (это вариант по умолчанию), то код, выполняющийся в нулевом кольце, может выполнять запись на любые страницы, даже на те, которые помечены «только для чтения».

64-битное кодирование

Весь 64-битный код использует плоскую модель памяти, поскольку в 64-битном режиме сегментация недоступна. Единственным исключением являются регистры FS и GS, которые по-прежнему добавляют свои базы.

Позиционная независимость кода в 64-битном режиме значительно проще, поскольку процессор напрямую поддерживает адресацию относительно RIP (ключевое слово REL).

На большинстве 64-битных платформ желательно сделать это значением по умолчанию, используя директиву DEFAULT REL.

64-битное программирование более-менее на 32-битное программирование, но указатели имеют длину 64 бита. Кроме того, все существующие платформы передают аргументы в регистрах, а не в стеке. Более того, 64-битные платформы по умолчанию используют SSE2 для операций с плавающей запятой. См. документацию по ABI для конкретной платформы.

64-битные платформы отличаются размерами основных типов данных C/C++ не только от 32-битных платформ, но и друг от друга. Если требуется тип данных определенного размера, лучше всего использовать типы, определенные в стандартном заголовке C **<inttypes.h>**.

Все известные 64-битные платформы, за исключением некоторых встроенных платформ, требуют, чтобы при входе в функцию стек был выровнен по 16 байтам. Чтобы обеспечить это, указатель стека (RSP) перед инструкцией CALL должен быть выровнен по нечетному кратному 8 байтам.

В 64-битном режиме размер инструкции по умолчанию по-прежнему составляет 32 бита. При загрузке значения в 32-битный регистр (но не в 8- или 16-битный) старшие 32 бита соответствующего 64-битного регистра обнуляются.

Имена регистров в режиме 64-бит

NASM использует следующие имена для регистров общего назначения в 64-битном режиме для 8-, 16-, 32- и 64-битных ссылок, соответственно:

AL/AH, CL/CH, DL/DH, BL/BH, **SPL, BPL, SIL, DIL, R8B-R15B**
AX, CX, DX, BX, SP, BP, SI, DI, **R8W-R15W**
EAX, ECX, EDX, EBX, ESP, EBP, ESI, EDI, **R8D-R15D**
RAX, RCX, RDX, RBX, RSP, RBP, RSI, RDI, R8-R15

Эти названия согласуются с документацией AMD и большинством других ассемблеров.

В документации Intel, однако, используются имена **R8L**-**R15L** для 8-битных ссылок на старшие регистры.

Эти имена можно использовать, определив их как макросы. Аналогично, если кто-то хочет использовать числовые имена для младших 8 регистров, можно их определить их как макросы. Для этого можно использовать стандартный пакет макросов **`altreg`** (LK5.2-Препроцессор NASM).

Непосредственные значения (Imm) и смещения в 64-битном режиме

В 64-битном режиме непосредственные значения и смещения обычно имеют ширину всего 32 бита. Поэтому NASM усекает большинство смещений и сводит их сразу к 32 битам.

Единственная инструкция, которая требует полного 64-битного непосредственные значения:

```
MOV reg64, imm64
```

NASM генерирует эту инструкцию всякий раз, когда программист использует MOV для загрузки непосредственного значения в 64-битный регистр.

Если это нежелательно, достаточно указать эквивалентный 32-битный регистр, который будет автоматически расширен процессором до нуля, либо указать непосредственное значение как DWORD:

```
mov rax, foo        ; 64-bit immediate                                  10
mov rax, qword foo  ; (identical)                                       10
mov eax, foo        ; 32-bit immediate, расширяется нулем в rax          5
mov rax, dword foo  ; 32-bit immediate, знаковое расширение в rax        7
```

Длина этих инструкций составляет 10, 10, 5 и 7 байт соответственно.

```
                            foo32 equ 0x12345678
                            foo64 equ 0x123456789abcdef0
00000000 48B8F0DEBC9A78563410           mov rax, foo64
0000000A 48B8F0DEBC9A78563410           mov rax, qword foo64
00000014 B878563412                     mov eax, foo32
00000019 48C7C078563412                 mov rax, dword foo32
```

Если включена оптимизация и NASM может определить во время своей работы, что будет достаточно более короткой инструкции, будет сгенерирована более короткая инструкция, если, конечно, не указано STRICT QWORD или STRICT DWORD (LK4-Ассемблер NASM), которые подавляют оптимизацию:

```
mov rax, 1              ; Assembles as "mov eax,1" (5 bytes)
mov rax, strict qword 1 ; Full 10-byte instruction
mov rax, strict dword 1 ; 7-byte instruction
mov rax, symbol         ; 10 bytes, not known at assembly time
lea rax, [rel symbol]   ; 7 bytes, usually preferred by the ABI
```

Листинг

```
00000020 B801000000                             mov rax, 1
00000025 48B8010000000000000                    mov rax, strict qword 1
0000002F 48C7C001000000                         mov rax, strict dword 1
00000036 48B8 00000038 [5100000000000000]       mov rax, .print
00000040 488D050A000000                         lea rax, [rel .print] ; 47 + 0A = 51
...
00000047 E805000000                             call .print           ; 4C + 5  = 51
0000004C E81C000000                             call .exit
                                      .print:
00000051 B801000000                             mov  rax, SYS_WRITE
```

Обратите внимание, что **`lea rax, [rel symbol]`** генерирует позиционно независимый код, тогда как инструкция **`mov rax, symbol`** генерирует позиционный.

Большинство ABI предпочитают или даже требуют позиционно-независимый код в 64-битном режиме. Однако инструкция MOV может ссылаться на символ в любом месте 64-битного адресного пространства, тогда как LEA может обращаться к символу только в пределах 2 ГБ от самой инструкции.

Единственные инструкции, которые принимают полное 64-битное смещение, — это загрузка или сохранение с использованием MOV регистров AL, AX, EAX, RAX (но не других регистров) по абсолютному 64-битному адресу.

Поскольку это относительно редко используемая инструкция (64-битный код обычно использует относительную адресацию), программист должен явно объявить размер смещения как ABS QWORD:

```
default abs

mov eax, [foo]            ; 32-bit absolute disp, sign-extended
mov eax, [a32 foo]        ; 32-bit absolute disp, zero-extended
mov eax, [qword foo]      ; 64-bit absolute disp

default rel

mov eax, [foo]            ; 32-bit relative disp
mov eax, [a32 foo]        ; d:o, address truncated to 32 bits(!)
mov eax, [qword foo]      ; error
mov eax, [abs qword foo]  ; 64-bit absolute disp
```

```
                          default abs

0000004F 8B0425[00000000]   mov eax, [foo] ; 32-bit abs. disp, sign-extended
00000056 67A1[00000000]     mov eax, [a32 foo] ;32-bit abs. disp, zero-exten
0000005C A10000005D [0000000000000000] mov eax, [qword foo] ; 64-bit abs disp

                          default rel
00000065 8B0595FFFFFF       mov eax, [foo]     ; 32-bit relative disp
0000006B 678B058EFFFFFF     mov eax, [a32 foo] ; d:o, address truncated to 32 bits!!
00000072 8B0588FFFFFF       mov eax, [qword foo]      ; error
         ****************** warning: displacement size ignored on absolute address
         ****************** warning: displacement size ignored on absolute address
00000078 A100000079 [0000000000000000] mov eax, [abs qword foo] ; 64-bit abs. disp
```

Абсолютное смещение со знакорасширенного диапазона может составлять от –2 ГБ до +2 ГБ.
Абсолютное смещение с нулевым расширением может иметь доступ от 0 до 4 ГБ.

Взаимодействие с 64-битными программами на C (Unix)

В Unix 64-битный ABI, а также x32 ABI (32-битный ABI с процессором в 64-битном режиме) определяется документами по адресу:

```
http://www.nasm.us/abi/unix64
```

Хотя они написаны для ассемблера с синтаксисом AT&T, эти концепции одинаково хорошо применимы и для ассемблирования в стиле NASM.

Ниже приводится упрощенное резюме.

Первые шесть целочисленных аргументов (слева) передаются в **RDI**, **RSI**, **RDX**, **RCX**, **R8** и **R9** (в указанном порядке). Дополнительные целочисленные аргументы передаются через стек.

Эти регистры, а также **RAX**, **R10** и **R11** портятся вызовами функций и, таким образом, доступны для использования функцией без сохранения.

Целочисленные возвращаемые значения передаются в RAX и RDX в указанном порядке.

Операции с плавающей запятой выполняются с использованием регистров SSE, за исключением **`long double`**, которое на большинстве платформ составляет 80 бит (TWORD).

!!! В Android **`long double`** составляет 64 бита и обрабатывается так же, как **`double`**.

Аргументы с плавающей запятой передаются в порядке от **XMM0** к **XMM7**.

Результаты возвращаются в **XMM0** и **XMM1**.

Значения **`long double`** передаются через стек и возвращаются в **ST0** и **ST1**.

Все регистры SSE и x87 уничтожаются вызовами функций.

В 64-битной системе Unix **`long`** — это 64 бита.

Целочисленные аргументы и аргументы регистра SSE рассматриваются отдельно, поэтому в случае

```
void foo(long a, double b, int c)
```

a передается в RDI;
b —// — в XMM0;
c —// — в ESI.

Взаимодействие с 64-битными программами на C (Win64)

Win64 ABI описан в документе по адресу:

```
http://www.nasm.us/abi/win64 → https://docs.microsoft.com/...MSDN...
```

Ниже приводится упрощенное резюме.

Первые четыре целочисленных аргумента передаются в **RCX**, **RDX**, **R8** и **R9** в указанном порядке. Дополнительные целочисленные аргументы передаются через стек.

Эти регистры, а также **RAX**, **R10** и **R11** портятся вызовами функций и, таким образом, доступны для использования функцией без сохранения.

Целочисленные возвращаемые значения передаются только в **RAX**.

Операции с плавающей запятой выполняются с использованием регистров SSE, за исключением **`long double`**.

Аргументы с плавающей запятой передаются от **XMM0** к **XMM3**;

Результаты возвращаются только в **XMM0**.

В Win64 тип **`long`** имеет размер в 32 бита;

`long long` или **`_int64`** — 64 бита.

Целочисленные аргументы и аргументы регистра SSE считаются вместе, поэтому в случае

```
void foo(long long a, double b, int c)
```

a передается в RCX;	RCX
b —//— в XMM1;	RDX, R8
c —//— в R8D.	R9