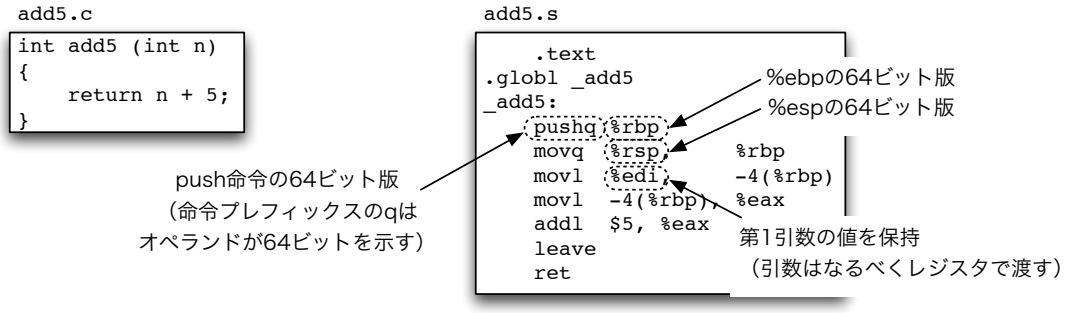
A.9　x86_64（64 ビットモード）の概要

- x86_64 は Intel64 アーキテクチャ [16] と AMD64 アーキテクチャ [17] への総称．i386[5] の自然な 64 ビット拡張として AMD 社が AMD64 を提案し，Intel 社が Intel64 として追随した．AMD64 と Intel64 は完全互換である．
- x86_64 は 2 つの動作モード：64 ビットモードと i386 互換モードを持つ．以下では 64 ビットモードを説明する．
- 64 ビットモードでは，デフォルトのオペランドサイズは 32 ビット，デフォルトのアドレスサイズは 64 ビット．

A.9.1　x86_64 の概要



- `add5.c` を `gcc -S -m64` でコンパイルすると `add5.s` になる（一部省略，結果はプラットフォーム依存）．これは x86_64 用のアセンブリコード．`-m64` オプションが未サポートのプラットフォームもある．`-m64` オプション無しで x86_64 用のアセンブリコードを出力するプラットフォームもある．
- 命令プレフィックスに `q` が追加された（例：push**q**）．これはオペランドのサイズが 64 ビットであることを示す．
- 汎用レジスタなどが 64 ビットに拡張された（例：`%rbp`，`rsp`）．`%rbp` はベースポインタ，`rsp` はスタックポインタをそれぞれ格納する．

A.9.2　x86_64 のリニアアドレスは 64 ビット長

- x86_64 のリニアアドレスは 64 ビット長．（i386 のリニアアドレスは 32 ビット長．）

A.9.3　x86_64 の命令サフィックス

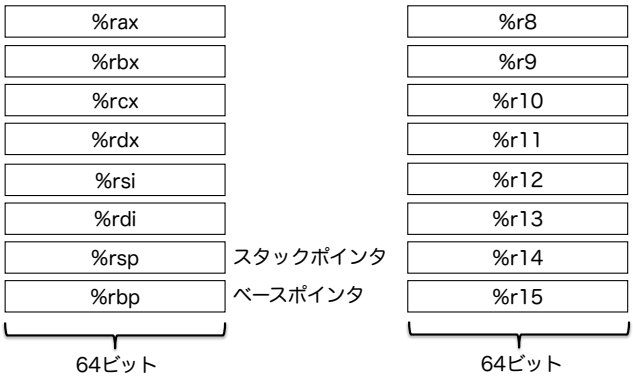
AT&T 形式	Intel 形式	意味
`q`	`QWORD PTR`	メモリオペランドのサイズは 8 バイト

AT&T 形式での例	Intel 形式での例
`movq $10, (%ebx)`	`mov QWORD PTR [ebx], 10`

- i386 の命令プレフィックス（`b`，`w`，`l`）（A.1.2 節（p.99））に加えて，x86_64 では `q` を使用する．`q` はオペランドサイズが 8 バイトであることを示す．
- ほとんどの命令に対して，32 ビットの命令（例：`movl`）と 64 ビットの命令（例：`movq`）の両方を使える．例外はスタック系の命令（`push`，`pop`，`call`，`ret`，`enter`，`leave`）であり，64 ビットの命令はあるが，32 ビットの命令は存在しない．64 ビットモードでは，例えば `pushl %eax` はエラーとなる．

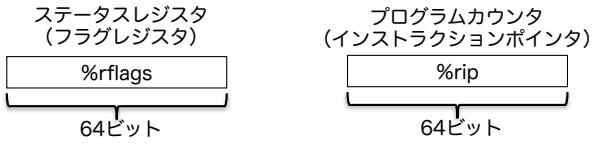
A.9.4 x86_64 のレジスタ

x86_64 の汎用レジスタ



- i386 の 8 つの 32 ビット汎用レジスタ（`%eax`，`%ebx`，`%ecx`，`%edx`，`%esi`，`%edi`，`%esp`，`%ebp`）は 64 ビットに拡張された（それぞれ，`%rax`，`%rbx`，`%rcx`，`%rdx`，`%rsi`，`%rdi`，`%rsp`，`%rbp`）．
- さらに 8 つの 64 ビット汎用レジスタが追加された（`%r8`，`%r9`，`%r10`，`%r11`，`%r12`，`%r13`，`%r14`，`%r15`）．

x86_64 のステータスレジスタとプログラムカウンタ：`%rflags`，`%rip`



- i386 の 32 ビットのステータスレジスタ `%eflags` は 64 ビットの `%rflags` に拡張された．下位 32 ビットは `%eflags` と同じ．上位 32 ビットは予約のため使用禁止．
- i386 の 32 ビットのプログラムカウンタ `%eip` は 64 ビットの `%rip` に拡張された．

x86_64 の caller-save/callee-save レジスタ [18, 19] （A.5.3 節（p.115）も参照）

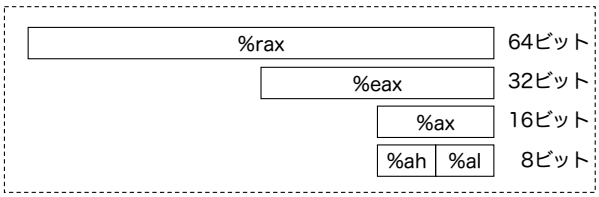
	汎用レジスタ
caller-save レジスタ	`%rax`，`%rcx`，`%rdx`，`%rsi`，`%rdi`，`%r8`〜`%r11`
callee-save レジスタ	`%rbx`，`%rbp`，`%rsp`，`%r12`〜`%r15`

x86_64 では第 6 引数まで引数をなるべく汎用レジスタで渡す [18, 19]

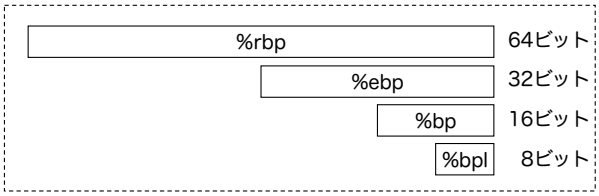
引数	レジスタ
第 1 引数	`%rdi`
第 2 引数	`%rsi`
第 3 引数	`%rdx`

引数	レジスタ
第 4 引数	`%rcx`
第 5 引数	`%r8`
第 6 引数	`%r9`

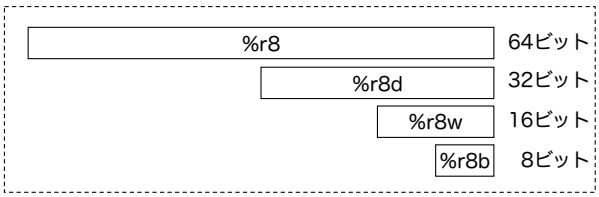
x86_64 のレジスタの別名



- `%rax` の別名（`%rbx`，`%rcx`，`%rdx` も同様）：
  - `%rax` の下位 32 ビットは `%eax` としてアクセス可能．
  - `%eax` の下位 16 ビットは `%ax` としてアクセス可能．
  - `%ax` の上位 8 ビットは `%ah` としてアクセス可能．
  - `%ax` の下位 8 ビットは `%al` としてアクセス可能．



- `%rbp` の別名（`%rsp`，`%rsi`，`%rdi` も同様）：
  - `%rbp` の下位 32 ビットは `%ebp` としてアクセス可能．
  - `%ebp` の下位 16 ビットは `%bp` としてアクセス可能．
  - `%bp` の下位 8 ビットは `%bpl` としてアクセス可能．



- `%r8` の別名（`%r9`〜`%r15` も同様）：
  - `%r8` の下位 32 ビットは `%r8d` としてアクセス可能．
  - `%r8d` の下位 16 ビットは `%r8w` としてアクセス可能．
  - `%r8w` の下位 8 ビットは `%r8b`（Intel 形式では `r8l`）としてアクセス可能．

x86_64 の新しいレジスタは `%ah`，`%bh`，`%ch`，`%dh` と一緒に使えない

- x86_64 の新しいレジスタは `%ah`，`%bh`，`%ch`，`%dh` と一緒に使えない．例えば，`movb %ah, (%r10)` や `movb %ah, %bpl` はエラーになる．
- 正確には，`rex` プレフィックス（A.9.5 節（p.126））付きの命令では，`%ah`，`%bh`，`%ch`，`%dh` を使えない．

A.9.5　x86_64 の rex プレフィックス

- rex プレフィックスは x86_64 で追加された命令プレフィックス（A.7 節（p.118））である．x86_64 の 64 ビットモードで，拡張レジスタや 64 ビットオペランドを使う場合にたいてい必要となる．
- rex プレフィックスは 1 バイト長である．下位 4 ビットが 4 つのフラグとして使われるため，rex プレフィックスの機械語コードは 40〜4F となる．

ニモニック	機械語コード	何の略か	動作
rex	40〜4F	64-bit extension	x86_64 の拡張レジスタや 64 ビットオペランドにアクセス

- GNU アセンブラが自動挿入するので，プログラマが明示的に rex プレフィックスを書く必要は通常は無い．
- rex プレフィックスの各ビットの意味は以下の通り（詳細は [17, 16] を参照）．rex64xyz が 4 つのフラグを全てセットした場合の rex プレフィックスのニモニック．64，x，y，z を省略することで，他の組み合わせも表現可能．

ビット位置	GNU アセンブラ表記	Intel マニュアル表記	意味
3	**64**	**REX.W**	オペランドサイズを 64 ビットにする
2	**x**	**REX.R**	ModR/M reg フィールド拡張を使用
1	**y**	**REX.X**	SIB index フィールド拡張を使用
0	**z**	**REX.B**	他のフィールド拡張を使用

A.9.6　x86_64 のニモニック

AT&T 形式	Intel 形式	動作
cltq	cdqe	%eax を %rax に（ロングをクワッドに）符号拡張
cqto	cqo	%rax を %rdx:%rax に（クワッドを 128 ビットに）符号拡張

AT&T 形式	Intel 形式	動作
movsbq	movsx	バイトをクワッドに符号拡張してデータ転送
movswq	movsx	ワードをクワッドに符号拡張してデータ転送
movslq	movsxd	ロングをクワッドに符号拡張してデータ転送
movzbq	movzx	バイトをクワッドにゼロ拡張してデータ転送
movzwq	movzx	ワードをクワッドにゼロ拡張してデータ転送

- x86_64 で，AT&T 形式と Intel 形式でニモニックが異なる機械語（i386 の場合は A.8.1 節（p.121）も参照）．
- movzlq（Intel 形式では movzxd）はなぜか存在しない．

A.9.7　x86_64 の演算

- 32 ビット演算の場合，32 ビットの計算結果をゼロ拡張して 64 ビットレジスタに格納する．
- 8 ビット演算や 16 ビット演算の場合，上位ビット（それぞれ 56 ビット，48 ビット）には変更無し．

コード例 A.9　レジスタの上位ビットがゼロになる場合・ならない場合

```
movq $0x1122334455667788, %rax
addl $1, %eax                    # %rax の上位 32 ビットはゼロになる
movq $0x1122334455667788, %rax
addw $1, %ax                     # %rax の上位 48 ビットに変化無し
movq $0x1122334455667788, %rax
addb $1, %al                     # %rax の上位 56 ビットに変化無し
```

A.9.8 x86_64 のアドレッシングモード

x86_64 の即値は 32 ビットまで

- x86_64 の機械語コード中の即値は（mov 命令を除いて）32 ビットまでで，即値は 64 ビットに拡張されていない．このため，値が-0x80000000〜0x7FFFFFFF の範囲（=32 ビット符号あり整数で表現できる範囲）を超える即値は指定できない．

コード例 A.10 即値が 32 ビットを超える場合（NG）・超えない場合（OK）

```
addl $0xFFFFFFFF, %eax          # OK
addq $0xFFFFFFFF, %rax          # NG  (0x7FFFFFFF を超えている)
addq $0xFFFFFFFFFFFFFFFF, %rax # OK  (0xFFFFFFFFFFFFFFFF=-0x1)
addq $0x7FFFFFF, %rax           # OK
addq $-0x80000000, %rax         # OK
addq $0xFFFFFFFF80000000, %rax # OK  (0xFFFFFFFF80000000=-0x80000000)
```

- 32 ビットの即値は 64 ビットの演算前に 64 ビットに符号拡張される．例えば，addq $-1, %eax 中の即値-1 は 32 ビットだが，加算の前に 64 ビット長に符号拡張される．
- mov 命令のみ，64 ビットの即値を指定できる．64 ビットの即値の使用を明示的に示すために movabs というニモニックを使用できる．

詳しい文法	例	例の動作
mov *imm64*, *r64*	movq $0x1122334455667788, %rax	%rax=0x1122334455667788
movabs *imm64*, *r64*	movabsq $0x1122334455667788, %rax	%rax=0x1122334455667788

コード例 A.11 move 命令のみ，64 ビットの即値を指定できる

```
movq    $0x1122334455667788, %rax # OK
movabsq $0x1122334455667788, %rax # OK
movq    $0x8000000000000000, %rax # OK
movabsq $0x8000000000000000, %rax # OK
movq    $1, %rax                  # OK  (即値は 32 ビット)
movabsq $1, %rax                  # OK  (即値は 64 ビット)
```

- 64 ビットのアドレスでジャンプするには，間接ジャンプを使う必要がある．例えば，jmp 0x1122334455667788 とは書けないため，movq $0x1122334455667788, %rax; jmp *%rax などと書く必要がある．

詳しい文法	例	例の動作
jmp *r/m64*	jmp *%rax	%rip+=%eax （near，絶対，間接ジャンプ）
	jmp *4(%rbp)	%rip+=*(%rbp+4) （near，絶対，間接ジャンプ）

x86_64 のラベルの扱い

- i386 と同様にラベルはアドレス定数．ただし，GNU アセンブラは x86_64 のラベルの扱いを文脈により変える．

コード例 A.12 文脈によるラベルの値の変化

```
addl _foo, %eax         # 32 ビット定数，絶対アドレス，64 ビットにゼロ拡張
addq _foo, %rax         # 32 ビット定数，絶対アドレス，64 ビットに符号拡張
movabsq _foo, %rax      # 64 ビット定数，絶対アドレス
addq _foo(%rip), %rax   # 32 ビット定数，相対アドレス（%rip+foo が，foo の絶対アドレスになる）
```

x86_64 のメモリ参照

- 64 ビットのメモリ参照には，section:disp(base, index, scale) と section:disp(%rip) の 2 つの形式を使用できる．どちらの場合も disp は符号あり 32 ビット定数であり，アドレス計算前に 64 ビット長に符号拡張される．
  - ◆ section:disp(base, index, scale) の形式では，disp は符号あり 32 ビット定数，base と index は 64 ビット汎用レジスタを指定可能（ただし index に %rsp は指定不可）．section と scale は i386 の 32 ビットのメモリ参照（A.1.4 節（p.102））と同じ．

disp		base		index		scale
		%rax		%rax		
		%rbx		%rbx		
		%rcx		%rcx		
		%rdx		%rdx		
		%rsp				1
8 ビット定数	+	%rbp	+	%rbp	×	2
32 ビット定数		%rsi		%rsi		4
		%rdi		%rdi		8
		%r8		%r8		
		⋮		⋮		
		%r15		%r15		

  - ◆ section:disp(%rip) の形式では，disp は符号あり 32 ビット定数，section は i386 の 32 ビットのメモリ参照（A.1.4 節（p.102））と同じ．(disp＋%rip) の計算結果をオフセット（セグメント先頭からの相対アドレス）として，メモリにアクセスする．この形式を **%rip 相対アドレッシング** %rip *relative addressing* という．

disp		
32 ビット符号あり定数	+	%rip

- x86_64 では，i386 の 32 ビットのメモリ参照（例：movq $1, 4(%ebp)）（A.1.4 節（p.102））も使用できる．この場合，32 ビットの disp は 64 ビットに**ゼロ拡張**される（cf. 64 ビットのメモリ参照では**符号拡張**される）．

x86_64 の disp（アドレス定数）は 32 ビットまで

- x86_64 のメモリ参照で使用する disp（アドレス定数）は（mov 命令を除いて）32 ビットまで．このため，値が -0x80000000〜0x7FFFFFFF の範囲（=32 ビット符号あり整数で表現できる範囲）を超える disp は指定できない．ただし，GNU アセンブラは 0x80000000〜0xFFFFFFFF の範囲の指定を自動的に-0x80000000〜-0x1 に変換する（この動作は即値の場合と異なっている）．

コード例 A.13 disp は（mov 命令を除いて）32 ビットまで

```
addq 0xFFFFFFFF,   %rax          # OK  (-0x1 と解釈)
addq 0xFFFFFFFFFFFFFFFF, %rax    # OK  (0xFFFFFFFFFFFFFFFF=-0x1)
addq 0x1FFFFFFFF,  %rax          # NG  (-0x80000000〜0x7FFFFFFF の範囲外)
addq 0x7FFFFFFF,   %rax          # OK
addq 0x80000000,   %rax          # OK  (-0x80000000 と解釈)
addq -0x80000000,  %rax          # OK
addq 0xFFFFFFFF80000000, %rax    # OK  (0xFFFFFFFF80000000=-0x80000000)
```

- 32 ビットの disp は 64 ビットのアドレス計算前に 64 ビットに符号拡張される（つまり 32 ビットの即値と同じ扱い）．
- mov 命令のみ，64 ビットの disp を指定できる．この場合，オペランドは以下の形式になる：ニモニックは movabs，メモリ参照は disp のみを指定する．base，index，scale は指定できない．他方のオペランドは %rax のみに限定．

詳しい文法	例	例の動作
movabs *moffs64*, %rax	movabsq 0x1122334455667788, %rax	%rax=*0x1122334455667788
movabs %rax, *moffs64*	movabsq %rax, 0x1122334455667788	*0x1122334455667788=%rax

moffs64 は 64 ビットの disp のみで指定するメモリ参照