x86/x86_64 関数呼び出し チートシート

2012-09-09 Rev0.1
Yoichi Imai <sunnyone41@gmail.com>

<x86>

(C言語ソース)

```
void 【CONVENTION】caller()
 { int result; result = callee(0xFFFF0001, 0xFFFF0002, 0xFFFF0003, 0xFFFF0004); }
int 【CONVENTION】callee(int arg1, int arg2, int arg3, int arg4)
 { int x, y; x = arg1 + arg2; x -= arg3; y = x * arg4; return y; }
```

【__cdecl】

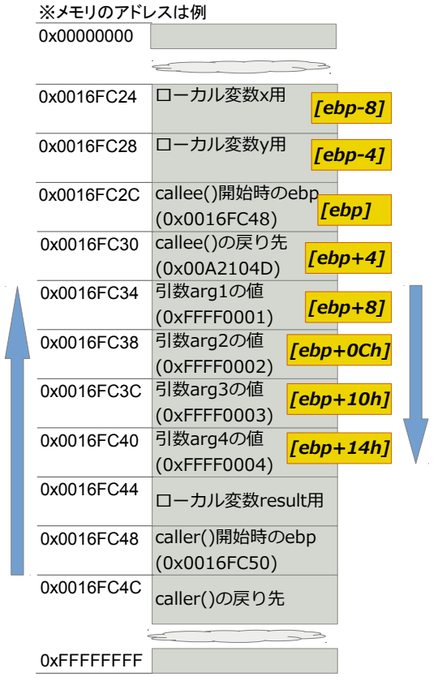
```
_caller:
00A21030   push ebp
    (ebpの値をスタックにプッシュ)
00A21031   mov  ebp,esp
    (スタックの基準=ベース(ebp)を今のスタックのトップ(esp)にする)
00A21033   push ecx
    (ecxをスタックに保存する=ローカル変数result)
00A21034   push 0FFFF0004h
    (arg4をスタックにpush)
00A21039   push 0FFFF0003h
    (arg3をスタックにpush)
00A2103E   push 0FFFF0002h
    (arg2をスタックにpush)
00A21043   push 0FFFF0001h
    (arg1をスタックにpush)
00A21048   call @ILT+0(_callee)
    (次のアドレス=00A2104Dをスタックに積んで呼び出し)


 -- callee()の呼び出し ---


00A2104D   add  esp,10h
    (arg1～4の分、スタックを低くする)
00A21050   mov  dword ptr [ebp-4],eax
    (eax=戻り値をローカル変数resultに入れる)
00A21053   mov  esp,ebp
00A21055   pop  ebp
00A21056   ret
```

eax ebx ecx edx
ebp esp edi esi

※メモリのアドレスは例




```
_callee:
00A21060   push ebp
    (ebpの値をプッシュ/espが0x16FC2Cに変化)
00A21061   mov  ebp,esp
    (上記のesp=0x16FC2Cがスタックのベースに)
00A21063   sub  esp,8
    (スタックを8バイト拡張(espが0x16FC24に))
00A21066   mov  eax,dword ptr [ebp+8]
    (eaxにarg1の値を入れる)
00A21069   add  eax,dword ptr [ebp+0Ch]
    (eaxにarg2の値を足す(eax = arg1 + arg2))
00A2106C   mov  dword ptr [ebp-8],eax
    (ローカル変数xにeaxの値を入れる)
00A2106F   mov  ecx,dword ptr [ebp-8]
    (ecxにローカル変数xの値を入れる)
00A21072   sub  ecx,dword ptr [ebp+10h]
    (ecxからarg3の値を引く (ecx = x - arg3))
00A21075   mov  dword ptr [ebp-8],ecx
    (ローカル変数xにecxの値を入れる)
00A21078   mov  edx,dword ptr [ebp-8]
    (edxにローカル変数xの値を入れる)
00A2107B   imul edx,dword ptr [ebp+14h]
    (edxにarg4の値をかける(edx = x * arg4)
00A2107F   mov  dword ptr [ebp-4],edx
    (ローカル変数yにedxの値を入れる)
00A21082   mov  eax,dword ptr [ebp-4]
    (eax=戻り値にローカル変数yの値を入れる)
00A21085   mov  esp,ebp
    (espをebpにする=呼び出し時のebpの場所に戻す)
00A21087   pop  ebp
    (呼び出し時にプッシュしたebpをとりだす)
00A21088   ret
```

【__stdcall】呼び出され側がスタッククリーンアップ

```
_caller:
  push ebp
  mov  ebp,esp
  push ecx
  push 0FFFF0004h
  push 0FFFF0003h
  push 0FFFF0002h
  push 0FFFF0001h
  call @ILT+5(_callee@16)
  mov  dword ptr [ebp-4],eax
  mov  esp,ebp
  pop  ebp
  ret
```

```
_callee@16:
  push ebp
  mov  ebp,esp
  sub  esp,8
  mov  eax,dword ptr [ebp+8]
  add  eax,dword ptr [ebp+0Ch]
  mov  dword ptr [ebp-8],eax
  mov  ecx,dword ptr [ebp-8]
  sub  ecx,dword ptr [ebp+10h]
  mov  dword ptr [ebp-8],ecx
  mov  edx,dword ptr [ebp-8]
  imul edx,dword ptr [ebp+14h]
  mov  dword ptr [ebp-4],edx
  mov  eax,dword ptr [ebp-4]
  mov  esp,ebp
  pop  ebp
  ret  10h
```

【__fastcall】呼び出され側がスタッククリーンアップ / 2つはレジスタ(ecx/edx)渡し

```
_caller:
  push ebp
  mov  ebp,esp
  push ecx
  push 0FFFF0004h
  push 0FFFF0003h
  mov  edx,0FFFF0002h
  mov  ecx,0FFFF0001h
  call @ILT+15(@callee@16)
  mov  dword ptr [ebp-4],eax
  mov  esp,ebp
  pop  ebp
  ret
```

```
@callee@16:
  push ebp
  mov  ebp,esp
  sub  esp,10h
  mov  dword ptr [ebp-10h],edx
  mov  dword ptr [ebp-0Ch],ecx
  mov  eax,dword ptr [ebp-0Ch]
  add  eax,dword ptr [ebp-10h]
  mov  dword ptr [ebp-8],eax
  mov  ecx,dword ptr [ebp-8]
  sub  ecx,dword ptr [ebp+8]
  mov  dword ptr [ebp-8],ecx
  mov  edx,dword ptr [ebp-8]
  imul edx,dword ptr [ebp+0Ch]
  mov  dword ptr [ebp-4],edx
  mov  eax,dword ptr [ebp-4]
  mov  esp,ebp
  pop  ebp
  ret  8
```

⇒GCC(Linux/x86)でも、スタックを大きく確保する / leave命令を利用する等
若干の違いはあるがほぼ同様

<x86_64>

(C言語ソース)

```
typedef long long int INT64;

INT64 callee(INT64 arg1, INT64 arg2, INT64 arg3,
             INT64 arg4, INT64 arg5, INT64 arg6,
             INT64 arg7, INT64 arg8);
void caller() {
   INT64 result;
   result = callee(0xAAAAFFFF0001, 0xAAAAFFFF0002,
                   0xAAAAFFFF0003, 0xAAAAFFFF0004,
                   0xAAAAFFFF0005, 0xAAAAFFFF0006,
                   0xAAAAFFFF0007, 0xAAAAFFFF0008);
}

INT64 callee(INT64 arg1, INT64 arg2,
             INT64 arg3, INT64 arg4,
             INT64 arg5, INT64 arg6,
             INT64 arg7, INT64 arg8)
{
  INT64 x1, x2, x3, x4;
  x1 = arg1 + arg2;
  x2 = arg3 - arg4;
  x3 = arg5 * arg6;
  x4 = arg7 / arg8;
  return x1 + x2 + x3 + x4;
}
```

x86環境：

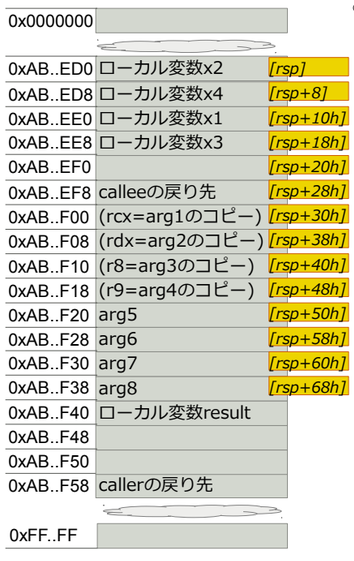
```
 cl.exe: Microsoft(R) 32-bit C/C++ Optimizing Compiler
    Version 16.00.30319.01 for 80x86
  gcc: gcc バージョン 4.6.3 (Ubuntu/Linaro 4.6.3-1ubuntu5)
```

x86_64環境：

```
 cl.exe: Microsoft (R) C/C++ Optimizing Compiler
    Version 16.00.30319.01 for x64
  gcc: gcc version 4.4.5 (Debian 4.4.5-8)
```

Microsoft 4つはレジスタ渡し(rcx/rdx/r8/r9) / 呼び出し側が32byte領域を確保

```
caller:
   sub  rsp,58h
   mov  rax,0AAAAFFFF0008h
   mov  qword ptr [rsp+38h],rax
   mov  rax,0AAAAFFFF0007h
   mov  qword ptr [rsp+30h],rax
   mov  rax,0AAAAFFFF0006h
   mov  qword ptr [rsp+28h],rax
   mov  rax,0AAAAFFFF0005h
   mov  qword ptr [rsp+20h],rax
   mov  r9,0AAAAFFFF0004h
   mov  r8,0AAAAFFFF0003h
   mov  rdx,0AAAAFFFF0002h
   mov  rcx,0AAAAFFFF0001h
   call @ILT+5(callee)
   mov  qword ptr [rsp+40h],rax
   add  rsp,58h
   ret
```




```
callee:
   mov  qword ptr [rsp+20h],r9
   mov  qword ptr [rsp+18h],r8
   mov  qword ptr [rsp+10h],rdx
   mov  qword ptr [rsp+8],rcx
   sub  rsp,28h ; 左図のrspはこれ以降
   mov  rax,qword ptr [rsp+38h]
   mov  rcx,qword ptr [rsp+30h]
   add  rcx,rax
   mov  rax,rcx
   mov  qword ptr [rsp+10h],rax
   mov  rax,qword ptr [rsp+48h]
   mov  rcx,qword ptr [rsp+40h]
   sub  rcx,rax
   mov  rax,rcx
   mov  qword ptr [rsp],rax
   mov  rax,qword ptr [rsp+50h]
   imul rax,qword ptr [rsp+58h]
   mov  qword ptr [rsp+18h],rax
   mov  rax,qword ptr [rsp+60h]
   cqo
   idiv rax,qword ptr [rsp+68h]
   mov  qword ptr [rsp+8],rax
   mov  rax,qword ptr [rsp]
   mov  rcx,qword ptr [rsp+10h]
   add  rcx,rax
   mov  rax,rcx
   add  rax,qword ptr [rsp+18h]
   add  rax,qword ptr [rsp+8]
   add  rsp,28h
   ret
```

rax rbx rcx rdx
rbp rsp rdi rsi
r8 r9 r10 r11
r12 r13 r14 r15

呼び出し側(caller)が保存
呼び出され側(callee)が保存

System V (GCC他) 6つはレジスタ渡し(rdi/rsi/rdx/rcx/r8/r9) / 128バイトのRed Zone

```
caller:
  push rbp
  mov  rbp,rsp
  sub  rsp,0x20
  mov  DWORD PTR [rsp+0x8],0xffff0008
  mov  DWORD PTR [rsp+0xc],0xaaaa
  mov  DWORD PTR [rsp],0xffff0007
  mov  DWORD PTR [rsp+0x4],0xaaaa
  mov  r9,0xaaaaffff0006
  mov  r8,0xaaaaffff0005
  mov  rcx,0xaaaaffff0004
  mov  rdx,0xaaaaffff0003
  mov  rsi,0xaaaaffff0002
  mov  rdi,0xaaaaffff0001
  call 400502 <callee>
  mov  QWORD PTR [rbp-0x8],rax
  leave
  ret
```

```
callee:
  push rbp
  mov  rbp,rsp
  mov  QWORD PTR [rbp-0x28],rdi
  mov  QWORD PTR [rbp-0x30],rsi
  mov  QWORD PTR [rbp-0x38],rdx
  mov  QWORD PTR [rbp-0x40],rcx
  mov  QWORD PTR [rbp-0x48],r8
  mov  QWORD PTR [rbp-0x50],r9
  mov  rax,QWORD PTR [rbp-0x30]
  mov  rdx,QWORD PTR [rbp-0x28]
  lea  rax,[rdx+rax*1]
  mov  QWORD PTR [rbp-0x20],rax
  mov  rax,QWORD PTR [rbp-0x40]
  mov  rdx,QWORD PTR [rbp-0x38]
  mov  rcx,rdx
  sub  rcx,rax
  mov  rax,rcx
  mov  QWORD PTR [rbp-0x18],rax
  mov  rax,QWORD PTR [rbp-0x48]
  imul rax,QWORD PTR [rbp-0x50]
  mov  QWORD PTR [rbp-0x10],rax
  mov  rax,QWORD PTR [rbp+0x10]
  mov  rdx,rax
  sar  rdx,0x3f
  idiv QWORD PTR [rbp+0x18]
  mov  QWORD PTR [rbp-0x8],rax
  mov  rax,QWORD PTR [rbp-0x18]
  mov  rdx,QWORD PTR [rbp-0x20]
  lea  rax,[rdx+rax*1]
  add  rax,QWORD PTR [rbp-0x10]
  add  rax,QWORD PTR [rbp-0x8]
  leave
  ret
```

rax rbx rcx rdx
rbp rsp rdi rsi
r8 r9 r10 r11
r12 r13 r14 r15