C Language VI—C/Assembly Interface

CMSC 313
Sections 01, 02

Linux/gcc/i386 Function Call Convention

Adapted from Richard Chang, CMSC 313 Spring 2013

Linux/gcc/i386 Function Call Convention

- There are many ways for code to call subroutines
- Steps involve delegation of responsibility for who will do what to stay out of the other's way
- Some factors to be considered:
  - Contention for registers
  - agreement about where parameters are passed
  - Support for language features, like recursion, variadic functions
  - help for debuggers
- These are ***conventions***, not hard rules

Adapted from Richard Chang, CMSC 313 Spring 2013

Linux/gcc/i386 Function Call Convention

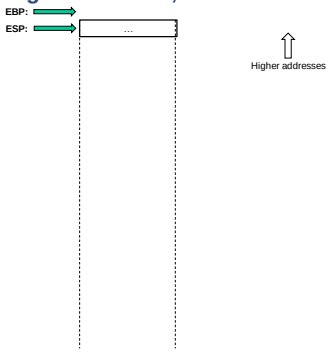
One standard convention is called "__cdecl":

- Parameters pushed *right to left* on the stack
  - So, first parameter is on top of the stack (i.e., lowest address)
- Caller saves EAX, ECX, EDX if needed
  - these registers will probably be used by the callee
- Callee saves EBX, ESI, EDI
  - there is a good chance that the callee does not need these
- EBP used as index register for parameters, local variables, and temporary storage
- Callee must restore caller's ESP and EBP
- Return value placed in EAX

Adapted from Richard Chang, CMSC 313 Spring 2013

__cdecl Calling Convention, Phase 1a

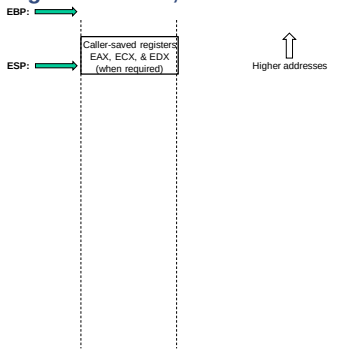
Phase 1a—executed by caller:



__cdecl Calling Convention, Phase 1a

Phase 1a—executed by caller:

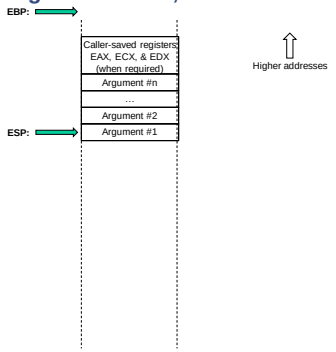
1. Push registers EAX, ECX, & EDX if it cares about values



__cdecl Calling Convention, Phase 1a

Phase 1a—executed by caller:

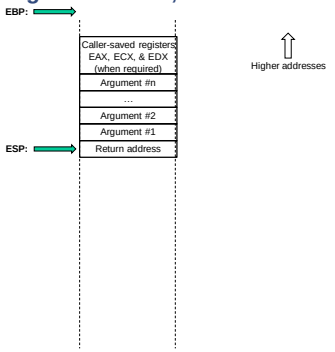
1. Push registers EAX, ECX, & EDX if it cares about values
2. Push function arguments onto stack *from right to left*



__cdecl Calling Convention, Phase 1a

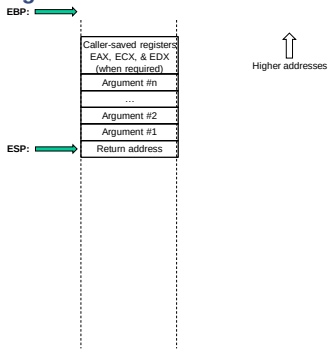
Phase 1a—executed by caller:

1. Push registers EAX, ECX, & EDX if it cares about values
2. Push function arguments onto stack *from right to left*
3. CALL the desired function



__cdecl Calling Convention—Phase 1b

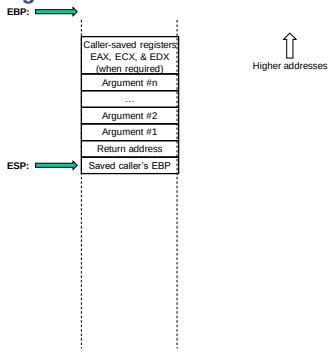
Phase 1b—executed by *callee:*



__cdecl Calling Convention—Phase 1b

Phase 1b—executed by *callee:*

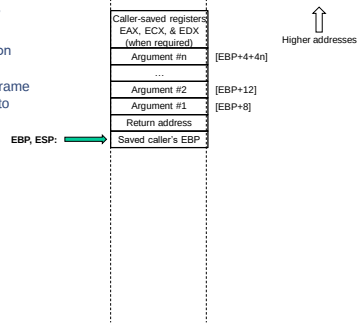
1. Save caller's EBP on stack



__cdecl Calling Convention—Phase 1b

Phase 1b—executed by *callee:*

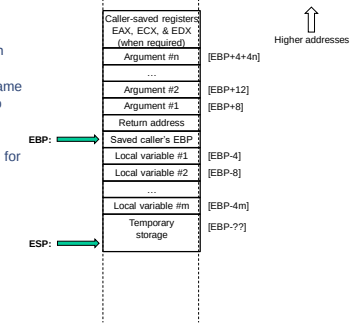
1. Save caller's EBP on stack
2. Replace with own frame (copy current ESP to EBP)



__cdecl Calling Convention—Phase 1b

Phase 1b—executed by *callee:*

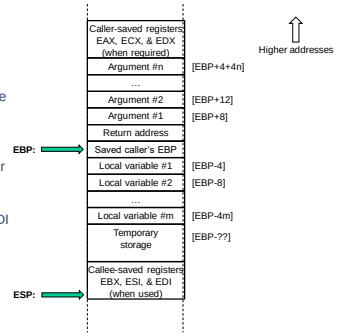
1. Save caller's EBP on stack
2. Replace with own frame (copy current ESP to EBP)
3. Subtract offset from ESP to create space for local and temporary storage



__cdecl Calling Convention—Phase 1b

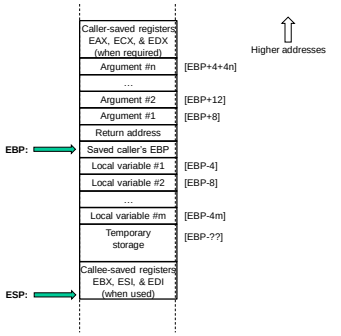
Phase 1b—executed by *callee:*

1. Save caller's EBP on stack
2. Replace with own frame (copy current ESP to EBP)
3. Subtract offset from ESP to create space for local and temporary storage
4. Save EBX, ESI, and EDI if it will use them



__cdecl Calling Convention—Phase 2

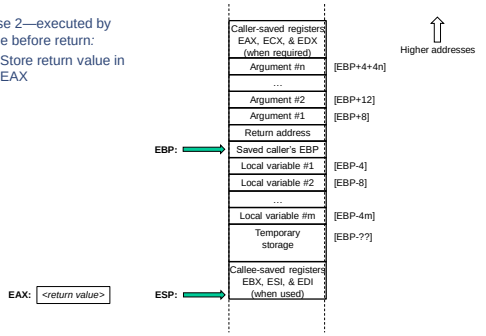
Phase 2—executed by callee before return:



__cdecl Calling Convention—Phase 2

Phase 2—executed by callee before return:

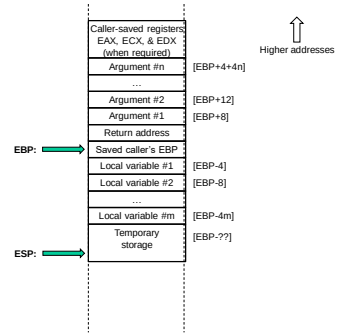
1. Store return value in EAX



__cdecl Calling Convention—Phase 2

Phase 2—executed by callee before return:

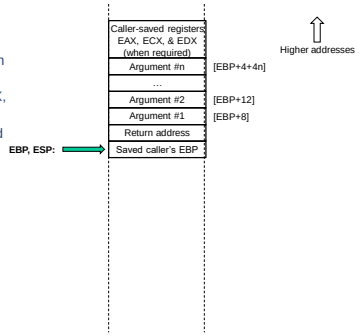
1. Store return value in EAX
2. Restore (POP) EBX, ESI, & EDI if used



__cdecl Calling Convention—Phase 2

Phase 2—executed by callee before return:

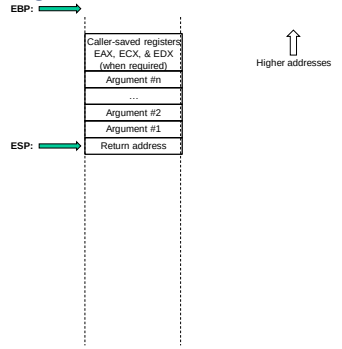
1. Store return value in EAX
2. Restore (POP) EBX, ESI, & EDI if used
3. Restore space used for local and temp (just copy EBP to ESP)



__cdecl Calling Convention—Phase 2

Phase 2—executed by callee before return:

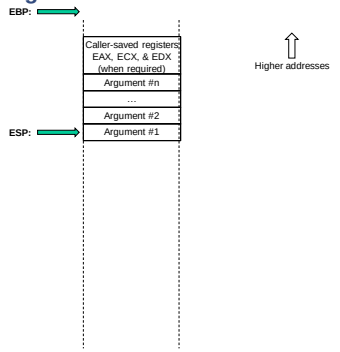
1. Store return value in EAX
2. Restore (POP) EBX, ESI, & EDI if used
3. Restore space used for local and temp (just copy EBP to ESP)
4. Restore (POP) caller's EBP



__cdecl Calling Convention—Phase 2a

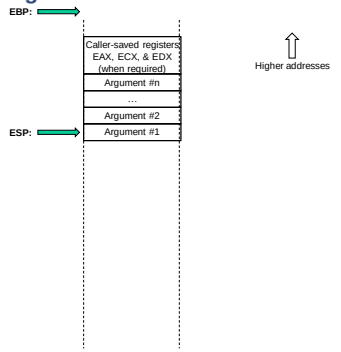
Phase 2a—executed by callee before return:

1. Store return value in EAX
2. Restore (POP) EBX, ESI, & EDI if used
3. Restore space used for local and temp (just copy EBP to ESP)
4. Restore (POP) caller's EBP
5. Return from function



__cdecl Calling Convention—Phase 2b

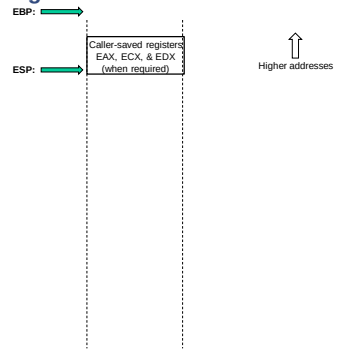
Phase 2b—executed by caller after return:



__cdecl Calling Convention—Phase 2b

Phase 2b—executed by caller after return:

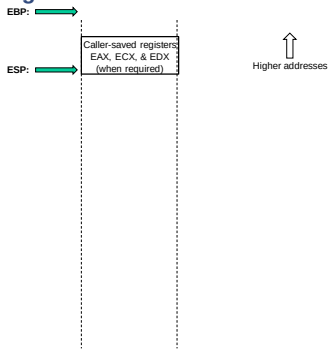
1. Remove arguments from stack (just add offset to ESP)



__cdecl Calling Convention—Phase 2b

Phase 2b—executed by caller after return:

1. Remove arguments from stack (just add offset to ESP)
2. Store return value in EAX away



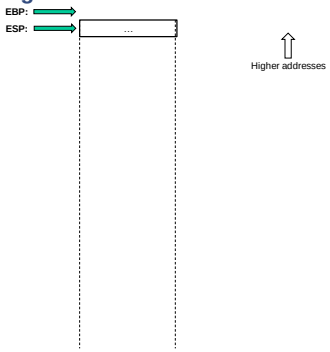
retval: <return value>

EAX: <return value>

__cdecl Calling Convention—Phase 2b

Phase 2b—executed by caller after return:

1. Remove arguments from stack (just add offset to ESP)
2. Store return value in EAX away
3. Restore values into EAX, ECX, & EDX if saved



retval: <return value>

EAX: <return value>

```
// File: cfunc.c
//
// Example of C function calls disassembled
//

#include <stdio.h>

// a silly function
//
int foo(int x, int y) {

   int z ;

   z = x + y ;
   return z ;
}

int main () {
   int b ;

   b = foo(35, 64) ;
   b = b + b ;
   printf ("b = %d\n", b) ;
}
linux3% gcc cfunc.c
linux3% a.out
b = 198
linux3%


linux3% gcc -S cfunc.c
linux3% i2g -g cfunc.s >cfunc.asm
linux3%
```

```
        .file   "cfunc.c"
        .version        "01.01"
gcc2_compiled.:
.text
        .align 4
.globl foo
        .type    foo,@function
foo:
        pushl %ebp
        movl %esp,%ebp
        subl $4,%esp
        movl 8(%ebp),%eax
        movl 12(%ebp),%edx
        leal (%edx,%eax),%ecx
        movl %ecx,-4(%ebp)
        movl -4(%ebp),%edx
        movl %edx,%eax
        jmp .L1
        .p2align 4,,7
.L1:
        leave
        ret
```

```
.Lfe1:
        .size    foo,.Lfe1-foo
.section        .rodata
.LC0:
        .string "b = %d\n"
.text
        .align 4
.globl main
        .type    main,@function
main:
        pushl %ebp
        movl %esp,%ebp
        subl $4,%esp
        pushl $64
        pushl $35
        call foo
        addl $8,%esp
        movl %eax,%eax
        movl %eax,-4(%ebp)
        movl -4(%ebp),%eax
        addl %eax,-4(%ebp)
        movl -4(%ebp),%eax
        pushl %eax
        pushl $.LC0
        call printf
        addl $8,%esp
.L2:
        leave
        ret
.Lfe2:
        .size    main,.Lfe2-main
        .ident   "GCC: (GNU) egcs-2.91.66 19990314/Linux (egcs-1.1.2
release)"
```

```
        ;FILE "cfunc.c"
gcc2_compiled.:
SECTION .text
        ALIGN 4
GLOBAL foo
        GLOBAL foo:function
foo:
        push  ebp
        mov  ebp,esp
        sub  esp,4
        mov  eax, [ebp+8]
        mov  edx, [ebp+12]
        lea  ecx, [edx+eax]
        mov  [ebp-4],ecx
        mov  edx, [ebp-4]
        mov  eax,edx
        jmp L1
        ;ALIGN 1<<4 ; IF < 7
L1:
        leave
        ret
```

```
.Lfe1:
        GLOBAL   foo:function (.Lfe1-foo)
SECTION         .rodata
.LC0:
        db      'b = %d',10,''
SECTION .text
        ALIGN 4
GLOBAL main
        GLOBAL main:function
main:
        push  ebp
        mov  ebp,esp
        sub  esp,4
        push  dword 64
        push  dword 35
        call foo
        add  esp,8
        mov  eax,eax
        mov  [ebp-4],eax
        mov  eax, [ebp-4]
        add  [ebp-4],eax
        mov  eax, [ebp-4]
        push  eax
        push  dword .LC0
        call printf
        add  esp,8
L2:
        leave
        ret
.Lfe2:
        GLOBAL   main:function (.Lfe2-main)
        :IDENT "GCC: (GNU) egcs-2.91.66 19990314/Linux (egcs-1.1.2
release)"
```

```
; File: printf1.asm
;
; Using C printf function to print
;
; Assemble using NASM:  nasm -f elf printf1.asm
;
; C-style main function.
; Link with gcc:  gcc printf1.o
;

; Declare some external functions
;
        extern printf                   ; the C function, we'll cal

        SECTION .data                   ; Data section

msg:    db "Hello, world: %c", 10, 0    ; The string to print.


        SECTION .text                   ; Code section.

        global main
main:
        push    ebp                     ; set up stack frame
        mov     ebp,esp

        push    dword 97                ; an 'a'
        push    dword msg               ; address of ctrl string
        call    printf                  ; Call C function
        add     esp, 8                  ; pop stack

        mov     esp, ebp                ; takedown stack frame
        pop     ebp                     ;   same as "leave" op

        ret
```

```
linux3% nasm -f elf printf1.asm
linux3% gcc printf1.o

linux3% a.out
Hello, world: a
linux3% exit
```

```
; File: printf2.asm
;
; Using C printf function to print
;
; Assemble using NASM:  nasm -f elf printf2.asm
;
; Assembler style main function.
; Link with gcc: gcc -nostartfiles printf2.asm
;

%define SYSCALL_EXIT  1

; Declare some external functions
;
        extern printf                   ; the C function, we'll cal

        SECTION .data                   ; Data section

msg:    db "Hello, world: %c", 10, 0    ; The string to print.


        SECTION .text                   ; Code section.

        global _start
_start:
        push    dword 97                ; an 'a'
        push    dword msg               ; address of ctrl string
        call    printf                  ; Call C function
        add     esp, 8                  ; pop stack

        mov     eax, SYSCALL_EXIT       ; Exit.
        mov     ebx, 0                  ; exit code, 0=normal
        int     0x80                    ; ask kernel to take over
```

```
linux3% nasm -f elf printf2.asm
linux3% gcc -nostartfiles printf2.o
linux3%

linux3% a.out
Hello, world: a
linux3%
```

```
// File: arraytest.c
//
// C program to test arrayinc.asm
//

void arrayinc(int A[], int n) ;

main() {

int A[7] = {2, 7, 19, 45, 3, 42, 9} ;
int i ;

   printf ("sizeof(int) = %d\n", sizeof(int)) ;

   printf("\nOriginal array:\n") ;
   for (i = 0 ; i < 7 ; i++) {
      printf("A[%d] = %d  ", i, A[i]) ;
   }
   printf("\n") ;

   arrayinc(A,7) ;

   printf("\nModified array:\n") ;
   for (i = 0 ; i < 7 ; i++) {
      printf("A[%d] = %d  ", i, A[i]) ;
   }
   printf("\n") ;

}
```

```
linux3% gcc -c arraytest.c
linux3% nasm -f elf arrayinc.asm
linux3% gcc arraytest.o arrayinc.o
linux3%
linux3% a.out
sizeof(int) = 4

Original array:
A[0] = 2  A[1] = 7  A[2] = 19  A[3] = 45  A[4] = 3  A[5] = 42  A[6] = 9

Modified array:
A[0] = 3  A[1] = 8  A[2] = 20  A[3] = 46  A[4] = 4  A[5] = 43  A[6] = 10
linux3%
```

```
; File: arrayinc.asm
;
; A subroutine to be called from C programs.
; Parameters: int A[], int n
; Result: A[0], ... A[n-1] are each incremented by 1


        SECTION .text
        global arrayinc

arrayinc:
        push    ebp                     ; set up stack frame
        mov     ebp, esp

        ; registers ebx, esi and edi must be saved, if used
        push    ebx
        push    edi

        mov     edi, [ebp+8]            ; get address of A
        mov     ecx, [ebp+12]           ; get num of elts
        mov     ebx, 0                  ; initialize count

for_loop:
        mov     eax, [edi+4*ebx]        ; get array element
        inc     eax                     ; add 1
        mov     [edi+4*ebx], eax        ; put it back
        inc     ebx                     ; update counter
        loop    for_loop

        pop     edi                     ; restore registers
        pop     ebx

        mov     esp, ebp                ; take down stack frame
        pop     ebp

        ret
```

```
// File: cfunc3.c
//
// Example of C function calls disassembled
// Return values with  more than 4 bytes
//

#include <stdio.h>

typedef struct {
   int part1, part2 ;
} stype ;


// a silly function
//
stype foo(stype r) {

   r.part1 += 4;
   r.part2 += 3 ;
   return r ;
}


int main () {
   stype r1, r2, r3  ;
   int n ;

   n = 17 ;
   r1.part1 = 74 ;
   r1.part2 = 75 ;
   r2.part1 = 84 ;
   r2.part2 = 85 ;
   r3.part1 = 93 ;
   r3.part2 = 99 ;

   r2 = foo(r1) ;

   printf ("r2.part1 = %d, r2.part2 = %d\n",
    r1.part1, r2.part2 ) ;

   n = foo(r3).part2 ;
}
```

```
        .FILE "cfunc3.c"
gcc2_compiled.:
SECTION .text
        ALIGN 4
GLOBAL foo
        GLOBAL foo:function
foo:                                    ; comments & spacing added
        push  ebp                       ; set up stack frame
        mov  ebp,esp

        mov  eax, [ebp+8]               ; addr to store return value
        add  dword [ebp+12],4           ; r.part1 = [ebp+12]
        add  dword [ebp+16],3           ; r.part2 = [ebp+16]

        ; return value
        ;
        mov  edx, [ebp+12]              ; get r.part1
        mov  ecx, [ebp+16]              ; get r.part2
        mov  [eax],edx                  ; put r.part1 in return valu
        mov  [eax+4],ecx                ; put r.part2 in return valu
        jmp L1
L1:
        mov  eax,eax                    ; does nothing
        leave                           ; bye-bye
        ret 4                           ; pop 4 bytes after return
.Lfe1:
```

```
        GLOBAL   foo:function (.Lfe1-foo)
SECTION         .rodata
.LC0:
        db      'r2.part1 = %d, r2.part2 = %d',10,''
SECTION .text
        ALIGN 4
GLOBAL main
        GLOBAL main:function
main:                                   ; comments & spacing added
        push  ebp                       ; set up stack frame
        mov  ebp,esp
        sub  esp,36                     ; space for local variables

        ; initialize variables
        ;
        mov  dword [ebp-28],17          ; n = [ebp-28]
        mov  dword [ebp-8],74           ; r1 = [ebp-8]
        mov  dword [ebp-4],75
        mov  dword [ebp-16],86          ; r2 = [ebp-16]
        mov  dword [ebp-12],85
        mov  dword [ebp-24],93          ; r3 = [ebp-24]
        mov  dword [ebp-20],99

        ; call foo
        ;
        lea  eax, [ebp-16]              ; get addr of r2
        mov  edx, [ebp-8]               ; get r1.part1
        mov  ecx, [ebp-4]               ; get r1.part2
        push  ecx                       ; push r1.part2
        push  edx                       ; push r1.part1
        push  eax                       ; push addr of r2
        call foo
        add  esp,8                      ; pop r1
                                        ; ret 4 popped r2's addr

        ; call printf
        ;
        mov  eax, [ebp-12]              ; get r2.part2
        push  eax                       ; push it
        mov  eax, [ebp-8]               ; get r2.part1
        push  eax                       ; push it
        push  dword .LC0                ; string constant's addr
        call printf
        add  esp,12                     ; pop off arguments
```

```
        ; call foo again
        ;
        lea  eax, [ebp-36]              ; addr of temp variable
        mov  edx, [ebp-24]              ; get r3.part1
        mov  ecx, [ebp-20]              ; get r3.part2
        push  ecx                       ; push r3.part2
        push  edx                       ; push r3.part1
        push  eax                       ; push addr of temp var
        call foo
        add  esp,8                      ; pop off arguments

        ; assign to n
        ;
        mov  eax, [ebp-32]              ; get part2 of temp var
        mov  [ebp-28],eax               ; store in n
L2:
        leave                           ; bye-bye
        ret
.Lfe2:
        GLOBAL   main:function (.Lfe2-main)
        .IDENT "GCC: (GNU) egcs-2.91.66 19990314/Linux (egcs-1.1.2
release)"
```