CMSC 313 Lecture 12

• Project 3 Questions
• How C functions pass parameters
• Project 4
Project 3: External Records

Due: Tue 10/07/03, Section 0101 (Chang) & Section 0301 (Macneil)
     Wed 10/08/03, Section 0201 (Patel & Bourner)

Objective

The objective of this programming project is to gain experience writing more complex assembly language programs and to use indexed addressing modes.

Assignment

Your assembly language program for this project will work with an externally defined array of records. This array is defined in a C program as follows:

```c
struct {
    char   realname[32] ;
    char   nickname[16] ;
    char   alignment[20] ;
    char   role[20] ;
    int    points ;
    int    level ;
} records[10] ;

int num_records = 10 ;
```

The records in the array have pre-initialized values not shown here. The full text of the C program is available on the GL file system at: /afs/umbc.edu/users/c/h/chang/pub/cs313/records.c

Your assembly language program must search through the array and find the record with the least number of points and the record with the alphabetically first nickname. It must then print out the realname field of these two records. E.g.,

Lowest Points: James Pressman
First Nickname: Dan Gannett

Implementation Notes

- The sample data in records.c contains 10 records, but your program should work with any number of records. The number of records is stored in the int variable num_records.
- In order to access the externally defined array and integer variable, you must have the following declaration in your assembly language program:
  ```asm
  extern records, num_records
  ```
- You must also make your own test cases. The example in records.c does not fully exercise your program. Your program will be graded based upon other test cases.
- You will need to link your assembly language program with the data defined in the C program:
  ```bash
  gcc -c records.c
  nasm -f elf report.asm
  ld records.o report.o
  ```
- An important part of this project is deciding how to use indexed addressing to access the data in the records. Think this through carefully. A clean and logical approach to this problem will yield clean and logical code that is easier to construct and, more importantly, easier to debug.
• Your program should be reasonably robust and report errors encountered (e.g., empty array) rather than crashing.

• Note that the strings stored in the array are C-style null-terminated strings.

• Nicknames should be compared using dictionary ordering. For example, any string starting with the letter 'a' comes before any string that starts with 'b'. In the case that one string is a prefix of another, the shorter string come first. E.g., "egg" comes before "egghead".

• To access each field of the record, you should use an offset from the address of the record. You should use %define constants instead of magic numbers. E.g.,

```c
#define NickOffset 32
#define AlignOffset 48
#define RoleOffset 68
#define PointsOffset 88
#define LevelOffset 92
#define RecSize 96
```

• Project 4 will be based upon Project 3, so keep in mind that you will need to extend/modify this program.

Turning in your program

Use the UNIX submit command on the GL system to turn in your project. You should submit at least 4 files: your assembly language program, at least 2 of your own test cases and a typescript file of sample runs of your program. The class name for submit is cs313_0101, cs313_0102 or cs313_0103 for respectively sections 0101 (Chang), 0201 (Patel & Bourner) or 0301 (Macneil). The name of the assignment name is proj3. The UNIX command to do this should look something like:

```bash
submit cs313_0103 proj3 report.asm myrec1.c myrec2.c typescript
```
Last Time

• **Stack Instructions: PUSH, POP**
  - PUSH adds an item to the top of the stack
  - POP removes an item from the top of the stack

• **Subroutine Instructions: CALL, RET**
  - CALL saves EIP on the stack and jumps to the subroutine
  - RET retrieves the caller’s EIP from the stack

• **Subroutine Examples**

UMBC, CMSC313, Richard Chang <chang@umbc.edu>
Linux/gcc/i386 Function Call Convention

- **Parameters pushed right to left on the stack**
  - first parameter on top of the stack

- **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**

UMBC, CMSC313, Richard Chang <chang@umbc.edu>
A typical stack frame for the function call:

```c
int foo (int arg1, int arg2, int arg3) ;
```

<table>
<thead>
<tr>
<th>ESP ==&gt;</th>
<th>.</th>
<th>.</th>
<th>.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Callee saved registers</td>
<td>EBX, ESI &amp; EDI</td>
<td>(as needed)</td>
<td></td>
</tr>
<tr>
<td>temporary storage</td>
<td>.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>local variable #2</td>
<td>[EBP - 8]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>local variable #1</td>
<td>[EBP - 4]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Caller's EBP</td>
<td>.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Return Address</td>
<td>.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Argument #1</td>
<td>[EBP + 8]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Argument #2</td>
<td>[EBP + 12]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Argument #3</td>
<td>[EBP + 16]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Caller saved registers</td>
<td>EAX, ECX &amp; EDX</td>
<td>(as needed)</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 1
The caller's actions before the function call

- Save EAX, ECX, EDX registers as needed
- Push arguments, last first
- CALL the function

<table>
<thead>
<tr>
<th>ESP ==&gt;</th>
<th>Return Address</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Arg #1 = 12</td>
</tr>
<tr>
<td></td>
<td>Arg #2 = 15</td>
</tr>
<tr>
<td></td>
<td>Arg #3 = 18</td>
</tr>
</tbody>
</table>

Caller saved registers EAX, ECX & EDX (as needed)

<table>
<thead>
<tr>
<th>EBP ==&gt;</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>.</td>
</tr>
</tbody>
</table>

Fig. 2
The callee's actions after function call

- Save main's EBP, set up own stack frame
  
  ```
  push ebp
  mov ebp, esp
  ```

- Allocate space for local variables and temporary storage

- Save EBX, ESI and EDI registers as needed
  
  EBP ==> main's EBP
  
  Return Address
  
  Arg #1 = 12
  
  Arg #2 = 15
  
  Arg #3 = 18
  
  Caller saved registers
  EAX, ECX, & EDX
  (as needed)

  ESP ==> Callee saved registers
  EBX, ESI, & EDI
  (as needed)

  temporary storage
  
  local variable #2
  
  local variable #1
  
  [EBP - 20]
  
  [EBP - 8]
  
  [EBP - 4]

  [EBP + 8]
  
  [EBP + 12]
  
  [EBP + 16]

  Fig. 4
The callee's actions before returning

- Store return value in EAX
- Restore EBX, ESI and EDI registers as needed
- Restore main's stack frame
  
  \[
  \begin{align*}
  \text{mov} & \quad \text{esp, ebp} \\
  \text{pop} & \quad \text{ebp}
  \end{align*}
  \]
- RET to main

<table>
<thead>
<tr>
<th>ESP ==&gt;</th>
<th>Arg #1 = 12</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Arg #2 = 15</td>
</tr>
<tr>
<td></td>
<td>Arg #3 = 18</td>
</tr>
</tbody>
</table>

Caller saved registers
EAX, ECX & EDX
(as needed)

<table>
<thead>
<tr>
<th>EBP ==&gt;</th>
<th>.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>.</td>
</tr>
</tbody>
</table>

Fig. 5
The caller's actions after returning

- POP arguments off the stack
- Store return value in EAX
- Restore EAX, ECX and EDX registers as needed

<table>
<thead>
<tr>
<th>Return Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arg #1 = 12</td>
</tr>
<tr>
<td>Arg #2 = 15</td>
</tr>
<tr>
<td>Arg #3 = 18</td>
</tr>
</tbody>
</table>

Caller saved registers EAX, ECX & EDX (as needed)

ESP ==> 

EBP ==> 

Fig. 6
// 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)"
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 call

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 call

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      080H ; 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 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:

Modified array:
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

```c
#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
",
            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 value
    mov [eax+4],ecx             ; put r.part2 in return value
    jmp L1
L1:                ; does nothing
    mov eax,eax                ; bye-bye
    leave
    ret 4                      ; pop 4 bytes after return
.Lfe1:
GLOBAL foo:function (.Lfe1-foo)

SECTION .rodata

.LC0:
\n  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],84 ; 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)"
Project 4: C Functions

Due: Tue 10/14/03, Section 0101 (Chang) & Section 0301 (Macneil)
      Wed 10/15/03, Section 0201 (Patel & Bourner)

Objective
The objective of this programming exercise is to practice writing assembly language programs that use the C function call conventions.

Assignment
Convert your assembly language program from Project 3 as follows:

1. Convert the program into one that follows the C function call convention, so it may be called from a C program. Your program should work with the following function prototype:
   void report (void *, unsigned int) ;

   The intention here is that the first parameter is a pointer to the records array and the second parameter has the number of items in that array.

2. Modify your program so it uses the strncmp() function from the C library to compare the nicknames of two records. The function prototype of strncmp() is:
   int strncmp(const char *s1, const char *s2, size_t n) ;

   The function returns an integer less than, equal to, or greater than zero if s1 (or the first n bytes thereof) is found, respectively, to be less than, to match, or be greater than s2.

3. Modify your program so that it prints out the entire record (not just the realname field) of the record with the least number of points and the record with the alphabetically first nickname. You must use the printf() function from the C library to produce this output. The output of your program would look something like:

   Lowest Points: James Pressman (jamieboy)
   Alignment: Lawful Neutral
   Role: Fighter
   Points: 57
   Level: 1
   First Nickname: Dan Gannett (danmeister)
   Alignment: True Neutral
   Role: Ranger
   Points: 7502
   Level: 3

A sample C program that should work with your assembly language implementation of the report() function is available on the GL file system: /afs/umbc.edu/users/c/h/chang/pub/cs313/records2.c

Implementation Notes

- Documentation for the printf() and strncmp() functions are available on the Unix system by typing man -S 3 printf and man -S 3 strncmp.

- Note that the strncmp() function takes 3 parameters, not 2. It is good programming practice to use strncmp() instead of strcmp() since this prevents runaway loops if the strings are not properly null terminated. The third argument should be 16, the length of the nickname field.
• As in Project 3, you must also make your own test cases. The example in records2.c does not fully exercise your program. As before, your program will be graded based upon other test cases. If you have good examples in Project 3, you can just reuse those.

• Use gcc to link and load your assembly language program with the C program. This way, gcc will call ld with the appropriate options:

  nasm -f elf report2.asm
gcc records2.c report2.o

• Notes on the C function call conventions are available on the web:

  http://www.csee.umbc.edu/~chang/cs313.f03/stack.shtml

• Your program should be reasonably robust and report errors encountered (e.g., empty array) rather than crashing.

**Turning in your program**

Use the UNIX submit command on the GL system to turn in your project. You should submit at least 4 files: your assembly language program, at least 2 of your own test cases and a typescript file of sample runs of your program. The class name for submit is cs313_0101, cs313_0102 or cs313_0103 for respectively sections 0101 (Chang), 0201 (Patel & Bourner) or 0301 (Macneil). The name of the assignment name is proj4. The UNIX command to do this should look something like:

submit cs313_0103 proj4 report2.asm myrec1.c myrec2.c typescript
Next Time

• Virtual Memory

• Cache Memory