Lecture Topics

- Today: Assembly Language Data Organization (H&H 6.3)
- Next: Exam #2

Announcements

- Self-study Module #9 (data organization)
- Project #8 (due no later than 6/20)
Exam #2

- Friday, 6/16 during lecture
- 20% of course grade
- One 8.5x11 note sheet (both sides) allowed
- Study suggestions on course website

Composite Data Objects

Besides scalar data objects, we also have composite data objects (more than one piece of data grouped together).

- Array – group of values, all with same type
- Record – group of values, may be different types
Data Structures

More complex data structures can be built out of records and arrays:

- Array of records
- Chain of records ("linked list")
- Tree of records ("binary search tree")

Example #25

Example which works with an array of records; each record contains a character string (array of type "char").

Function “main” written in C and function “print_all” written in assembly.

~cse320/Examples/example25.pdf
Data for one student: name (character string) and two exam scores (integers):

```c
struct student
{
    char    name[12];
    short   exam1;
    short   exam2;
};
```

Record mapping table for `struct student`:

<table>
<thead>
<tr>
<th>field</th>
<th>size of field</th>
<th>offset in record</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>12 * 1 byte</td>
<td>+0</td>
</tr>
<tr>
<td>exam1</td>
<td>2 bytes</td>
<td>+12</td>
</tr>
<tr>
<td>exam2</td>
<td>2 bytes</td>
<td>+14</td>
</tr>
</tbody>
</table>

We need a total of 16 bytes for one record (no padding required).
The compiler will allocate 16 bytes when it processes the declaration of an object of type `struct student`:

```
+0 ─── name (12 bytes)
+12 ─── exam1 (2 bytes)
+14 ─── exam2 (2 bytes)
```

```bash
<1 lemon:~/Examples > cat example25.driver.c

#include <stdio.h>
#include "/user/cse320/lib/memlib.h"
#define MAX 8

struct student
{
    char name[12];
    short exam1;
    short exam2;
};

void print_all( struct student *, int );
```
int main()
{
    struct student list[MAX];
    int i;

    for (i=0; i<MAX; i++)
    {
        scanf("%s %d %d", &list[i].name[0],
               &list[i].exam1, &list[i].exam2);
    }
    print_all( &list[0], MAX );

    display( &list[0], MAX*16, 16 );
}

<2 lemon:/Examples > gcc example25.driver.c \
    example25.support.s /user/cse320/lib/memlib.o

<3 lemon:/Examples > a.out < example25.data

Student name  Exam 1  Exam 2
-------------  ------  ------
Brown         70      75
Doe           74      84
Evans         49      57
Fernandez     89      72
Gray          66      77
Green         85      92
Jones         81      83
Smith         50      62
Memory contents from 7e955974 to 7e9559f4

7e955974: 42 72 6f 77 6e 00 00 00 00 00 00 00 46 00 4b 00
7e955974: 44 6f 65 00 00 00 00 00 cc af f2 76 4a 00 54 00
7e955994: 45 76 61 6e 73 00 00 00 00 00 00 00 31 00 39 00
7e9559a4: 46 65 72 6e 61 6e 64 65 7a 00 00 00 59 00 48 00
7e9559b4: 47 72 61 79 00 00 00 e8 4b d9 76 42 00 4d 00
7e9559c4: 47 72 65 65 6e 00 95 7e 8c 51 f0 76 55 00 5c 00
7e9559d4: 4a 6f 6e 65 73 00 01 00 c8 07 01 00 51 00 53 00
7e9559e4: 53 6d 69 74 68 00 00 00 7c 03 01 00 32 00 3e 00

<4 lemon:~/Examples > cat example25.support.s

    .global print_all
    .text
    .balign 4
print_all:
push    {r4,r5,r6,r7,r8,lr}

    mov    r4, r0          @ 1st arg: base address
    mov    r5, r1          @ 2nd arg: num records
    ldr    r0, =fmt1       @ print column headers
    bl    printf

14
mov r6, #0          @ init index to zero

loop:
    cmp r6, r5          @ comp index, elements
    bge endloop

    lsl r7, r6, #4      @ offset <= index * 16
    add r8, r4, r7      @ addr <= base + offset

    ldr r0, =fmt2       @ print one record
    add r1, r8, #0      @ address of name field
    ldrh r2, [r8, #12]  @ contents exam1 field
    ldrh r3, [r8, #14]  @ contents exam2 field
    bl printf

    add r6, r6, #1      @ increment index by one
    b loop

endloop:

    ldr r0, =fmt3       @ print a blank line
    bl printf

    pop {r4,r5,r6,r7,r8,lr}
    bx lr

fmt1:  .ascii "\n"
       .ascii "Student name Exam 1 Exam 2\n"
       .asciz "------------ ------ ------\n"

fmt2:  .asciz "%-12s %6d %6d\n"

fmt3:  .asciz "\n"
Example #25 recap

Array of 8 elements, each element 16 bytes.

Access to individual elements (one record):

```assembly
lsl    r7, r6, #4    @ offset <= index * 16
add    r8, r4, r7    @ addr <= base + offset

@ r8 contains address of current record
```

Access to fields within one record:

```assembly
ldr    r0, =fmt2    @ print one record
add    r1, r8, #0    @ address of name field
ldrh   r2, [r8, #12]    @ contents exam1 field
ldrh   r3, [r8, #14]    @ contents exam2 field
bl     printf
```

@ r8 contains address of current record
Example #26

Example which works with a chain of records; each record contains a character string (array of type “char”).

Function “main” written in C and function “print_all” written in assembly.

~cse320/Examples/example26.pdf

Record for Example #26

Data for one student: name (character string) and two exam scores (integers), as well as pointer:

```c
struct student
{
    char name[12];
    short exam1;
    short exam2;
    struct student *next;
};
```
Record mapping table for `struct student`:

<table>
<thead>
<tr>
<th>field</th>
<th>size of field</th>
<th>offset in record</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>12 * 1 byte</td>
<td>+0</td>
</tr>
<tr>
<td>exam1</td>
<td>2 bytes</td>
<td>+12</td>
</tr>
<tr>
<td>exam2</td>
<td>2 bytes</td>
<td>+14</td>
</tr>
<tr>
<td>next</td>
<td>4 bytes</td>
<td>+16</td>
</tr>
</tbody>
</table>

We need a total of 20 bytes for one record (no padding required).

Chain of records

N user nodes, "dummy" node at head of chain
#include <stdio.h>
#include <stdlib.h>
#define MAX 8

struct student
{
    char name[12];
    short exam1;
    short exam2;
    struct student *next;
};

void print_all( struct student * );

int main()
{
    struct student *item, *head, *tail;
    int i;

    head = (struct student *)
        malloc( sizeof( struct student ) );

    head->next = NULL;

    tail = head;
for (i=0; i<MAX; i++)
{
    item = (struct student *)
        malloc( sizeof( struct student ) );

    item->next = NULL;

    scanf( "%s %hd %hd", &item->name[0],
            &item->exam1, &item->exam2 );

    tail->next = item;
    tail = item;
}
print_all( head->next );
NULL = 0

.global print_all

.text
.balign 4
print_all:
    push   {r4,lr}
    mov    r4, r0            @ addr of first record
    ldr    r0, =fmt1        @ print column headers
    bl     printf

loop:
    cmp    r4, #NULL        @ compare addr and NULL
    beq    endloop

    @ print contents of one record
    @ (remaining code on next slide)

    ldr    r4, [r4, #+16]   @ retrieve address of
                          @ the next record
    b      loop

endloop:
@ print first part of one record

    ldr r0, =fmt2
    mov r1, r4          @ addr current record
    add r2, r4, #0      @ addr name field
    bl printf

@ print second part of one record

    ldr r0, =fmt3
    ldrh r1, [r4, #+12] @ exam1 field
    ldrh r2, [r4, #+14] @ exam2 field
    ldr r3, [r4, #+16] @ addr next record
    bl printf

ldr r0, =fmt4       @ print a blank line
bl printf

pop {r4,lr}
bx lr

fmt1: .ascii "\n"
   .ascii "Current .... \n"
   .asciz "-------- .... \n"

fmt2: .asciz "%08x %-12s "
fmt3: .asciz "%6d %6d %08x\n"

fmt4: .asciz "\n"
   .balign 4
Example #26 recap

Chain of 8 nodes (plus dummy node at head of chain), each node 20 bytes.

Access to individual elements:

@ r4 contains address of current record

```
ldr    r4, [r4, #+16]  @ address next record
```

Example #26 recap

Access to fields within one record:

@ r4 contains address of current record

```
ldr    r0, =fmt2
mov    r1, r4          @ addr current record
add    r2, r4, #0      @ addr name field
bl     printf
ldr    r0, =fmt3
ldrh   r1, [r4, #+12]  @ exam1 field
ldrh   r2, [r4, #+14]  @ exam2 field
ldr    r3, [r4, #+16]  @ addr next record
bl     printf
```
Recall the conventions for passing parameters to a function:

- up to four arguments passed in R0-R3
- additional arguments passed in the stack

Let’s look at an example where there are more than four arguments.

Example #27

Function "main" written in C, calls function "sum" with six arguments.

Function "sum" written in ARM assembly language, computes sum of the six arguments.

~cse320/Examples/example27.pdf
```c
#include <stdio.h>

int sum(int, int, int, int, int, int);

int main()
{
    int result;

    result = sum(0x11, 0x22, 0x33, 0x44, 0x55, 0x66);

    printf("\nSum: %#x\n", result);
}
```

```assembly
.global sum
.text
.balign 4
sum:
push  {lr}

// calculate sum of six arguments

pop   {lr}
bx    lr
```
sum:

    push {lr}

    add    r0, r0, r1
    add    r0, r0, r2
    add    r0, r0, r3

    ldr    r3, [sp, #4]
    add    r0, r0, r3

    ldr    r3, [sp, #8]
    add    r0, r0, r3

    pop    {lr}
    bx     lr

<3 lemon:~/Examples > gcc example27.driver.c \  
                        example27.support.s

<4 lemon:~/Examples > a.out

Sum: 0x165
Example #27 recap

First four args in R0-R3, last two args in stack

@ Get 5th argument from stack

     ldr r3, [sp, #4]
     add r0, r0, r3

@ Get 6th argument from stack

     ldr r3, [sp, #8]
     add r0, r0, r3

Example #27 recap

Why sp+4 and sp+8?

sum:

    push {lr}

Contents of stack after push {lr}:

    7ed93a34:  00010480
    7ed93a38:  00000055
    7ed93a3c:  00000066
Example #28

Function "main" written in assembly language, calls function "sum" with six arguments.

Function "sum" written in C, computes sum of the six arguments.

~cse320/Examples/example28.pdf

```
<1 lemon:~/Examples > cat example28.driver.s

.global main
.text
.balign 4
main:
push   {lr}

// call function sum, display result

pop    {lr}
bx      lr
```
mov    r0, #0x66
sub    sp, sp, #4
str    r0, [sp]

mov    r0, #0x55
sub    sp, sp, #4
str    r0, [sp]

mov    r3, #0x44
mov    r2, #0x33
mov    r1, #0x22
mov    r0, #0x11
bl     sum

add    sp, sp, #8

mov    r1, r0
ldr    r0, =fmt
bl     printf

pop    {lr}
bx     lr

fmt:   .asciz  "\nSum: %#x\n"
       .balign 4
<2 lemon:~:/Examples > cat example28.support.c

int sum( int, int, int, int, int, int );

int sum( int a, int b, int c, int d, int e, int f )
{
    return a+b+c+d+e+f;
}

<3 lemon:~:/Examples > gcc example28.driver.s \
                    example28.support.c

<4 lemon:~:/Examples > a.out

Sum: 0x165
Example #28 recap

First four args in R0-R3, last two args in stack

@@ Put 6th argument in stack

mov r0, #0x66
sub sp, sp, #4
str r0, [sp]

@@ Put 5th argument in stack

mov r0, #0x55
sub sp, sp, #4
str r0, [sp]

Example #28 recap

Function "main" (the caller) must delete the arguments from the stack after function "sum" (the callee) returns:

mov r3, #0x44
mov r2, #0x33
mov r1, #0x22
mov r0, #0x11
bl sum

add sp, sp, #8