Course: CMPE244

Name: Sarika Satish Natu
ID: 014529138(EE)

Lab: #1
Date: 9/4/2020

Struct Addr
Packed and Padded structures.

Source code screenshots:

```c
typedef struct {
  float f1; // 4 bytes
  char c1;  // 1 byte
  float f2;
  char c2;
} my_s_padd /*__attribute__((packed))*/;

typedef struct {
  float f1; // 4 bytes
  float f2;
  char c2;
  char c1; // 1 byte
} my_s_padd_des /*__attribute__((packed))*/;

typedef struct {
  char c2;
  char c1;  // 1 byte
  float f1; // 4 bytes
  float f2;
} my_s_padd_asc /*__attribute__((packed))*/;

typedef struct __attribute__((packed)) {
  float f1; // 4 bytes
  char c1;  // 1 byte
  float f2;
  char c2;
} my_s_pack;

#pragma pack(push, 1)
typedef struct {
  float f1; // 4 bytes
  float f2;
  char c2;
  char c1; // 1 byte
} my_s_pack_pragma;
#pragma pack(pop)

my_s_padd s_padd;
my_s_padd_des s_padd_des;
my_s_padd_asc *s_padd_asc;
my_s_pack s_pack;
my_s_pack *s_pack_ptr;
my_s_pack_pragma s_pack_pragma;
```

```
puts("Starting RTOS");
printf("\nSize of padded structure with element order not changed: %d bytes\n"
       "floats 0x%p 0x%p\n"
       "chars  0x%p 0x%p\n",
       sizeof(s_padd), &s_padd.f1, &s_padd.f2, &s_padd.c1, &s_padd.c2);

printf("\nSize of padded structure with elements in ascending order; "
       "accessing structure elements using pointer: %d bytes\n"
       "floats 0x%p 0x%p\n"
       "chars  0x%p 0x%p\n",
       sizeof(*s_padd_asc), &s_padd_asc->f1, &s_padd_asc->f2, &s_padd_asc->c1, &s_padd_asc->c2);

printf("\nSize of padded structure with elements in descending order"
       ": %d bytes\n"
       "floats 0x%p 0x%p\n"
       "chars  0x%p 0x%p\n",
       sizeof(s_padd_des), &s_padd_des.f1, &s_padd_des.f2, &s_padd_des.c1, &s_padd_des.c2);

printf("\nSize of packed structure with element order not changed; "
       "using __attribute__((packed)): %d bytes\n"
       "floats 0x%p 0x%p\n"
       "chars  0x%p 0x%p\n",
       sizeof(s_pack), &s_pack.f1, &s_pack.f2, &s_pack.c1, &s_pack.c2);

printf("\nSize of packed structure with element order not changed; "
       "using __attribute__((packed))and pointer: %d bytes\n"
       "floats 0x%p 0x%p\n"
       "chars  0x%p 0x%p\n",
       sizeof(*s_pack_ptr), &s_pack_ptr->f1, &s_pack_ptr->f2, &s_pack_ptr->c1, &s_pack_ptr->c2);

printf("\nSize of packed structure with elements in descending order;"
       " using #pragma pack(push,1) and #pragma pack(pop): %d bytes\n"
       "floats 0x%p 0x%p\n"
       "chars  0x%p 0x%p\n",
       sizeof(s_pack_pragma), &s_pack_pragma.f1, &s_pack_pragma.f2, &s_pack_pragma.c1, &s_pack_pragma.c2);
```

Telemetry output:

```
--------------------------------------------------------------------------------
peripherals_init(): Low level startup
WARNING: SD card could not be mounted

I2C slave detected at address: 0x38
I2C slave detected at address: 0x62
I2C slave detected at address: 0x72

entry_point(): Entering main()
Starting RTOS

Size of padded structure with element order not changed: 16 bytes
floats 0x0x10001314 0x0x1000131c
chars  0x0x10001318 0x0x10001320

Size of padded structure with elements in ascending order; accessing structure elements using pointer: 12 bytes
floats 0x0x4 0x0x8
chars  0x0x1 0x0

Size of padded structure with elements in descending order: 12 bytes
floats 0x0x1000132c 0x0x10001330
chars  0x0x10001335 0x0x10001334

Size of packed structure with element order not changed; using __attribute__((packed)): 10 bytes
floats 0x0x10001308 0x0x1000130d
chars  0x0x1000130c 0x0x10001311

Size of packed structure with element order not changed; using __attribute__((packed))and pointer: 10 bytes
floats 0x0 0x0x5
chars  0x0x4 0x0x9

Size of packed structure with elements in descending order; using #pragma pack(push,1) and #pragma pack(pop): 10 bytes
floats 0x0x10001338 0x0x1000133c
chars  0x0x10001341 0x0x10001340

List of commands (use help <name> to get full help if you see ...):
            crash : Deliberately crashes the system to demonstrate how ...
              i2c : i2c read 0xDD 0xRR <n>...
         tasklist : Outputs list of RTOS tasks, CPU and stack usage....
--------------------------------------------------------------------------------
```

Assumption: Computer architecture is 32 bits i.e. 4 bytes word size.
Explanation:

1. Structure my_s_padd → Order of the structure elements is not changed
   As structure is unpacked, padding is used by default.
   Here as structure elements are not declared in a specific order wrt size, the padding inserts many unused bytes and increases the structure size.

   Total bytes = 16 bytes

4 bytes	f1			
4 bytes	c1	padding	padding	padding
4 bytes	f2			
4 bytes	c2	padding	padding	padding

   0x10001314(Address of f1) + 4 bytes = 0x10001318(Address of c1) + 4 bytes = 0x1000131c(Address of f2) + 4 bytes = 0x10001320(Address of c2)(4 bytes)

2. Structure my_s_padd_asc → Order of the structure elements is placed in ascending order of size and structure elements are accessed using pointer
   As structure is unpacked, padding is used by default.
   Here as structure elements is declared in a specific order wrt size, the padding is done in a better way. It is a good practice to do so.

Total bytes = 12 bytes

4 bytes	c2	c1	padding	padding
4 bytes	f1			
4 bytes	f2			

3. Structure my_s_padd_des → Order of the structure elements is placed in descending order of size
   As structure is unpacked, padding is used by default.
   Here as structure elements is declared in a specific order wrt size, the padding is done in a better way. It is a good practice to do so.

   Total bytes = 12 bytes

4 bytes	f1			
4 bytes	f2			
4 bytes	c2	c1	padding	padding

   0x1000132c(Address of f1) + 4 bytes = 0x10001330(Address of f2) + 4 bytes = 0x10001334(Address of c2) + 1 bytes = 0x10001335(Address of c1)(1 byte)

4. Structure my_s_pack → Order of the structure elements is not changed and structure elements are accessed using pointer
   As structure is packed, padding is not used.
   Structure is packed using __attribute__((packed)).
   Here the structure is packed and unpacked per 1 byte. So padding is not done and less memory is required by the structure.

   Total bytes = 10 bytes

4 bytes	f1 – byte 1	f1 – byte 2	f1 – byte 3	f1 – byte 4
4 bytes	c1	c2	f2 – byte 1	f2 – byte 2
2 bytes	f2 – byte 3	f2 – byte 4		

5. Structure my_s_pack_pragma → Order of the structure elements in descending order of size
   As structure is packed, padding is not used.
   Structure is packed using #pragma pack(push, 1) and #pragma pack(pop)
   Here the structure is packed and unpacked (pushed and popped) per n (here 1) byte. So padding is not done, and less memory is required by the structure.

   Total bytes = 10 bytes

4 bytes	f1 – byte 1	f1 – byte 2	f1 – byte 3	f1 – byte 4
4 bytes	f2 – byte 1	f2 – byte 2	f2 – byte 3	f2 – byte 4
2 bytes	c2	c1		

   0x10001338(Address of f1) + 4 bytes = 0x1000133c(Address of f2) + 4 bytes = 0x10001340(Address of c2) + 1 bytes = 0x10001341(Address of c1)(1 byte)