test.c

```c
#include <stdio.h>
#include <stdlib.h>

typedef enum _myNums {
    zero, one, two, three
} myNums;



typedef struct _myStruct1 {
    char c1;
    char c2;
    char c3;
    char c4;
    int i;
} myStruct1;
typedef struct _myStruct2 {
    char c1;
    char c2;
    int i;
    char c3;
    char c4;
} myStruct2;


typedef union _myUnion1{
    int i;
    char c1;
    char c2;
    char c3;
    char c4;
} myUnion1;
typedef union _myUnion2{
    int i;
    struct struct_t{
        char c1;
        char c2;
        char c3;
        char c4;
    } Struct_t;
    struct struct_t a;
} myUnion2;


int arr[10][5][2] = {1,2,3};
myStruct1 struct1 = {.i = 10, .c1 = 'a', .c2 = 'b'};
myStruct2 struct2 = {.i = 20, .c1 = 'a', .c2 = 'b'};
enum _myNums en1 = one;
myUnion1 union1 = {.i = 30, .c1 = 'a', .c2 = 'b'};
myUnion2 union2 = {.i = 0x89ABCDEF};
float myFloat = 1.5;
double myDouble = 1.5;
double *myPointer;
```

```c
int main(){
    //3D array
    arr[5][1][0] = 4;
    arr[5][1][1] = 5;
    arr[5][2][0] = 6;

    //Record (Struct)
    struct1.i = 100;
    struct1.c1 = 'a';
    struct1.c2= 'b';
    struct1.c3 = 'c';
    struct1.c4 = 'd';
    struct2.i = 200;
    struct2.c1 = 'x';
    struct2.c2= 'y';
    struct2.c3 = 'z';
    struct2.c4 = 'w';

    //Enumeration

    printf("%d\n", en1);
    for (int i = one ; i <= three; i++){
        printf("%d\n", i);
    }
```

```c
    //Union
    printf("%x %x %x %x %x\n", union1.i, union1.c1, union1.c2, union1.c3, union1.c4);
    printf("%x %x %x %x %x\n", union2.i, union2.a.c1, union2.a.c2, union2.a.c3, union2.a.c4);

    union2.a.c1 = 0x01;
    union2.a.c2 = 0x23;
    union2.a.c3 = 0x45;
    union2.a.c4 = 0x67;
    printf("%x %x %x %x %x\n", union2.i, union2.a.c1, union2.a.c2, union2.a.c3, union2.a.c4);

    //Float
    myFloat += 0.5;

    //Pointer
    myPointer = &myDouble;
    printf("%lf\n", *myPointer);
    return 0;
}
```

test.s

```
    .file   "test.c"
    .globl  arr
    .data
    .align 32
    .type   arr, @object
    .size   arr, 400
arr:
    .long   1
    .long   2
    .long   3
    .zero   4
    .zero   24
    .zero   360
    .globl  struct1
    .align 8
    .type   struct1, @object
    .size   struct1, 8
struct1:
    .byte   97
    .byte   98
    .zero   2
    .long   10
    .globl  struct2
    .align 8
    .type   struct2, @object
    .size   struct2, 12
struct2:
    .byte   97
    .byte   98
    .zero   2
    .long   20
    .zero   4
    .globl  en1
    .align 4
    .type   en1, @object
    .size   en1, 4
en1:
    .long   1
    .globl  union1
    .align 4
    .type   union1, @object
    .size   union1, 4
union1:
    .byte   98
    .zero   3
    .globl  union2
    .align 4
    .type   union2, @object
    .size   union2, 4
union2:
    .long   -1985229329
    .globl  myFloat
    .align 4
    .type   myFloat, @object
    .size   myFloat, 4
myFloat:
    .long   1069547520
    .globl  myDouble
    .align 8
    .type   myDouble, @object
    .size   myDouble, 8
myDouble:
    .long   0
    .long   1073217536
    .comm   myPointer,8,8
    .section    .rodata
.LC0:
    .string "%d\n"
.LC1:
    .string "%x %x %x %x %x\n"
```

```
.LC3:
    .string "%lf\n"
    .text
    .globl  main
    .type   main, @function
main:
.LFB2:
    .cfi_startproc
    pushq   %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq    %rsp, %rbp
    .cfi_def_cfa_register 6
    subq    $32, %rsp
    movl    $4, arr+208(%rip)
    movl    $5, arr+212(%rip)
    movl    $6, arr+216(%rip)
    movl    $100, struct1+4(%rip)
    movb    $97, struct1(%rip)
    movb    $98, struct1+1(%rip)
    movb    $99, struct1+2(%rip)
    movb    $100, struct1+3(%rip)
    movl    $200, struct2+4(%rip)
    movb    $120, struct2(%rip)
    movb    $121, struct2+1(%rip)
    movb    $122, struct2+8(%rip)
    movb    $119, struct2+9(%rip)
    movl    en1(%rip), %eax
    movl    %eax, %esi
    movl    $.LC0, %edi
    movl    $0, %eax
    call    printf
    movl    $1, -4(%rbp)
    jmp .L2
.L3:
    movl    -4(%rbp), %eax
    movl    %eax, %esi
    movl    $.LC0, %edi
    movl    $0, %eax
    call    printf
    addl    $1, -4(%rbp)
.L2:
    cmpl    $3, -4(%rbp)
    jle .L3
    movzbl  union1(%rip), %eax
    movsbl  %al, %edi
    movzbl  union1(%rip), %eax
    movsbl  %al, %esi
    movzbl  union1(%rip), %eax
    movsbl  %al, %ecx
    movzbl  union1(%rip), %eax
    movsbl  %al, %edx
    movl    union1(%rip), %eax
    movl    %edi, %r9d
    movl    %esi, %r8d
    movl    %eax, %esi
    movl    $.LC1, %edi
    movl    $0, %eax
    call    printf
    movzbl  union2+3(%rip), %eax
    movsbl  %al, %edi
    movzbl  union2+2(%rip), %eax
    movsbl  %al, %esi
    movzbl  union2+1(%rip), %eax
    movsbl  %al, %ecx
    movzbl  union2(%rip), %eax
    movsbl  %al, %edx
    movl    union2(%rip), %eax
    movl    %edi, %r9d
    movl    %esi, %r8d
    movl    %eax, %esi
    movl    $.LC1, %edi
```

```
  movl    $0, %eax
  call    printf
  movb    $1, union2(%rip)
  movb    $35, union2+1(%rip)
  movb    $69, union2+2(%rip)
  movb    $103, union2+3(%rip)
  movzbl  union2+3(%rip), %eax
  movsbl  %al, %edi
  movzbl  union2+2(%rip), %eax
  movsbl  %al, %esi
  movzbl  union2+1(%rip), %eax
  movsbl  %al, %ecx
  movzbl  union2(%rip), %eax
  movsbl  %al, %edx
  movl    union2(%rip), %eax
  movl    %edi, %r9d
  movl    %esi, %r8d
  movl    %eax, %esi
  movl    $.LC1, %edi
  movl    $0, %eax
  call    printf
  movss   myFloat(%rip), %xmm1
  movss   .LC2(%rip), %xmm0
  addss   %xmm1, %xmm0
  movss   %xmm0, myFloat(%rip)
  movq    $myDouble, myPointer(%rip)
  movq    myPointer(%rip), %rax
  movq    (%rax), %rax
  movq    %rax, -24(%rbp)
  movsd   -24(%rbp), %xmm0
  movl    $.LC3, %edi
  movl    $1, %eax
  call    printf
  movl    $0, %eax
  leave
  .cfi_def_cfa 7, 8
  ret
    .cfi_endproc
.LFE2:
    .size   main, .-main
    .section    .rodata
    .align 4
.LC2:
    .long   1056964608
    .ident  "GCC: (Ubuntu 5.4.0-6ubuntu1~16.04.12) 5.4.0 20160609"
    .section    .note.GNU-stack,"",@progbits
```

20181210 유창호

Global 에서 선언된 부분 해석

1. 3D Array (변수 이름 arr)

global 위치에 arr 이름의 크기 400 byte, 32 byte alignment를 한 공간을 할당한다. 내부의 시작 주소에 long 크기(4byte)의 1을 저장하고 4byte 뒤 주소에 long 2를, 또 4byte 뒤에 long 3을 저장한다. 배열의 나머지 공간들 (4+24+360 byte) 에는 0을 채운다.

2. Record Struct (변수 이름 struct1, struct2)

global 위치에 struct1 이름의 크기 8byte, 8byte alignment를 한 공간을 할당한다. 내부의 시작 주소에 byte 크기의 97('a')를 저장하고 1byte 뒤에 98('b')를 저장한다. 그 뒤의 2byte는 초기화 되지 않아서 0으로 채운다. 마지막 long (4byte) 크기에 10을 저장한다.

global 위치에 struct2 이름의 크기 12byte, 8byte alignment를 한 공간을 할당한다. 내부의 시작 주소에 byte 크기의 97('a')를 저장하고 1byte 뒤에 98('b')를 저장한다. 그 다음 요소인 long 20을 저장할 때는 4byte alignment를 사용해서 2byte를 0으로 채운 뒤에 long 20을 저장한다. 그다음 4byte는 초기화되지 않은 c3, c4와 0으로 채운다. struct1과 비교했을 때 요소의 순서의 align을 하는 과정에서 최적화 되지 않아 4byte를 손해보았다.

3. Enumerator (변수 이름 en1)

global 위치에 en1 이름의 크기 4byte, 4byte alignment를 한 공간을 할당한다. long 1 값을 초기화한다.

4. Union (변수 이름 union1, union2)

global 위치에 union1 이름의 크기 4byte, 4byte align을 해서 공간을 할당한다. union은 요소들이 같은 공간을 쓰기 때문에 가장 마지막으로 초기화된 c2='b'에 따라 byte 98('b')를 저장한다. 나머지 3byte는 0으로 채운다.

global 위치에 union2 이름의 크기 4byte, 4byte align을 해서 공간을 할당한다. i=0x89ABCDEF로 초기화되었기 때문에 2's compliment 로 표현한 long -1985229329 값을 저장한다.

5. Float (변수 이름 myFloat, myDouble)

global 위치에 4byte 크기의 4byte alignment를 한 myFloat 공간을 할당한다. 1.5 값으로 초기화 하기 위해서 long 1069547520 을 저장했는데 이는 single precision 에 따른 것이다. $(-1)^S \times 2^{e-bias} \times (M)$ 공식에 의해 S=0 e=127 M=0.5가 나오고 이것을 4byte에서 표현하면 $\underbrace{0}_{S}\underbrace{01111111}_{e}\underbrace{10\cdots0}_{M}{}_{(2)}$, 0x3FC00000 즉 $1069547520_{(10)}$ 이다.

global 위치에 8byte 크기의 8byte alignment 한 myDouble 공간을 할당한다. 1.5 값으로 초기화하기 위해서 long 1073217536을 저장했는데 이는 double precision에 따른 것이다. $(-1)^S \times 2^{e-bias} \times (M)$ 에 의해 S=0 e=1023 M=0.5 가 나오고 이것을 8byte에 표현하면 $\underbrace{0}_{S}\underbrace{01111111111}_{e}\underbrace{10\cdots0}_{M}$, 0x3FF80000 00000000 이다. 하지만 double은 8byte 가 메모리에 올라갈 때 뒤 4byte를 저장하고 (long 0) 그 뒤에 long 1073217536 을 저장한다. double은 두 개의 long 을 사용해서 저장한다는 것을 알 수 있었다.

6. Pointer (변수 이름 myPointer)

global 위치에 8byte 크기의 8byte alignment를 적용한 myPointer를 할당한다

main 에서 실행되는 부분 해석

1. 3D Array (arr[1][5][2])

arr[5][1][0] = 4 → movl $4, arr+208(%rip) 이 대응된다. 계산 원리는 $(5\times(5\cdot2)+1\times(2)+0)\cdot 4byte(int) = 208$

arr[5][1][1] = 5 → movl $5, arr+212(%rip) $(5\times(5\cdot2)+1\times(2)+1)\cdot 4byte = 212$

arr[5][2][0] = 6 → movl $6, arr+216(%rip) $(5\times(5\cdot2)+2\times(2)+0)\cdot 4byte = 216$ 이다.

여기서 row major는 쓰는 것을 알 수 있는데, arr[5][1][1] 뒤의 4byte는 arr[5][2][0] 의 주소다.

2. Record Struct (struct1, struct2)

struct1은 요소들의 순서가 {char: c1, char: c2, char: c3, char: c4, int: i} 이기에 i, c1, c2, c3, c4 순으로 값을 재할당할 때 struct1+4(%rip), struct1, struct1+1(%rip), struct1+2(%rip), struct+3(%rip)로 된다.

반면 struct2의 요소들은 순서가 {char: c1, char: c2, int: i, char: c3, char: c4}이기 때문에 i, c1, c2, c3, c4 순으로 값을 재할당할 때 struct2+4(%rip), struct2, struct2+(%rip), struct2+8(%rip), struct2+9(%rip)로 된다. 요소의 순서상 c2 ~ i 사이에 2byte padding이 있고 c4 뒤에도 2byte padding이 있기 때문에 4byte alignment를 지키기 위해 struct1보다 크기에서 4byte 손해를 보고 있다.

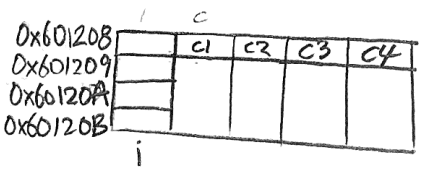
3. Enumerator (en1)

Enum MyNums {zero, one, two, three} 인 상황에서 en1은 one 값인 1로 초기화된 것을 볼 수 있었다.

for 문에서 i = one ; i<= three ; i++ 를 수행할 때 addl $1, -4(%rbp) 를 수행하며 cmp $3, -4(%rbp) 조건을 만족할 때까지 확인하고 있다. (3 = three)

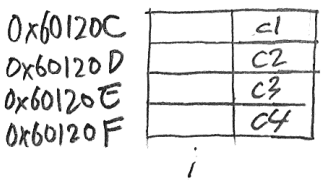
4. Union (union1, union2)

union1의 저장 상태는 다음과 같다



union1.i, union1.c1, union1.c2, union1.c3, union.c4 를 출력하는 부분이 동일하게 union1(%rip) 값을 참조하고 있다.
i, c1, c2, c3, c4 의 시작 주소가 모두 union1 주소를 가리킨다.

반면 union2의 저장상태는 다음과 같다



union2는 {int i, struct a{c1, c2, c3, c4}} 로 정의되었기에 4byte i, 4byte struct의 union 이다.

c1 : union2(%rip) c2 : union2+1(%rip) c3 : union2+2(%rip) c4 : union2+3(%rip) 에 대응된다.

또한 c1 = 0x01 c2 = 0x23 c3 = 0x45 c4 = 0x67 로 할당할 때 i값이 0x67452301 이 되는 것을 보면서 little endian을 쓰는 것을 알 수 있다.

5. Float (myFloat)

1.5 값을 가진 myFloat에 0.5를 더했다. myFloat 값을 %xmm1 레지스터에 저장하고 .LC2(%rip) 위치의 0.5 값을 %xmm0 레지스터에 옮겨서 더한 값을 myFloat(%rip)에 저장했다. 이때 .LC2 위치의 값은 1056964608 인데 이전에 앞서 살펴보았듯 single precision으로 나타낸 0.5 였다. $(-1)^S \times 2^{E-bias} \times (M)$ 일 때 0.5를 표현하려면 S=0 E=126 M=0 즉 00111111000...0 = 0x3F000000 = 1056964608이다.

6. Pointer (myPointer)

movq $myDouble, myPointer(%rip) 로 myDouble 값을 myPointer에 저장한다.

myPointer를 역참조 할 때는 movq myPointer(%rip), %rax 로 rax 레지스터에 주소를 저장하고 movq (%rax), %rax 로 주소를 역참조한다.