# SPARC V8
# Memory Alignment 실습

인하대학교 컴퓨터공학과

12224422 오세훈

# INDEX

# OVERVIEW

본 내용은 「**SPARC 환경 실시간 운영체제 RTEMS 실습**」 교재를 기반으로,
**GDB를 활용하여 SPARC 아키텍처를 보다 심층적으로 이해하고자 하는 목적**으로 작성되었습니다.

RTEMS 실습 과정에서 실질적으로 도움이 되는 주요 내용만을 중심으로 구성하였으며,
실습에 직접 활용하기에 적합한 핵심 사항 위주로 다루고 있습니다.
따라서 SPARC 아키텍처 및 RTEMS 전반에 대한 보다 상세한 설명과 이론적 배경은
해당 교재를 참고해 주시기 바랍니다.

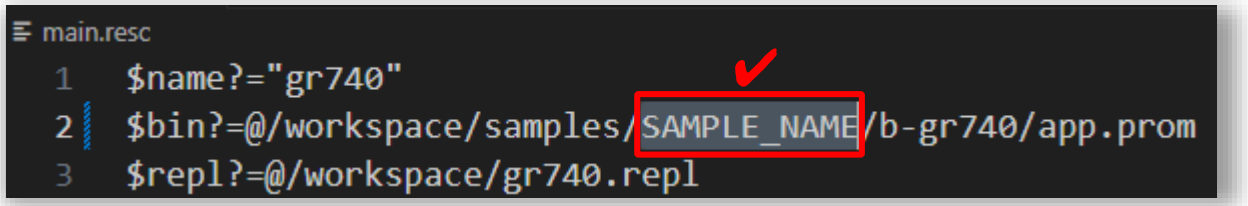
본 자료가 SPARC 기반 시스템의 동작 원리를 이해하고
디버깅 역량을 향상시키는 데 유용한 참고 자료가 되기를 바랍니다.

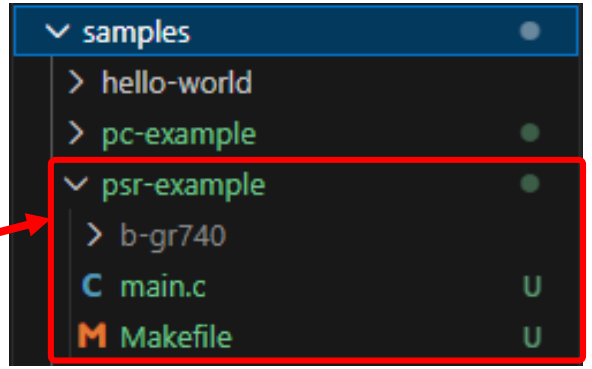
# OVERVIEW

- **샘플 코드 적용하는 방법**
  - 샘플 코드 다운로드 (원하는 위치에서 cmd 또는 powershell 열기)

```
$ git clone https://github.com/kjude22/SPARC-V8-Samples.git
```

  - /workspace/samples 폴더에 샘플 폴더를 복사
    - 개별 샘플 폴더를 **main.resc 경로에 맞게 /samples 상위**에 넣도록
  - /workspace/main.resc 에서 경로 변경




**"Drag here to copy"**

※ 자세한 내용 「**Renode GR740 설치 및 디버깅 환경설정**」 참고

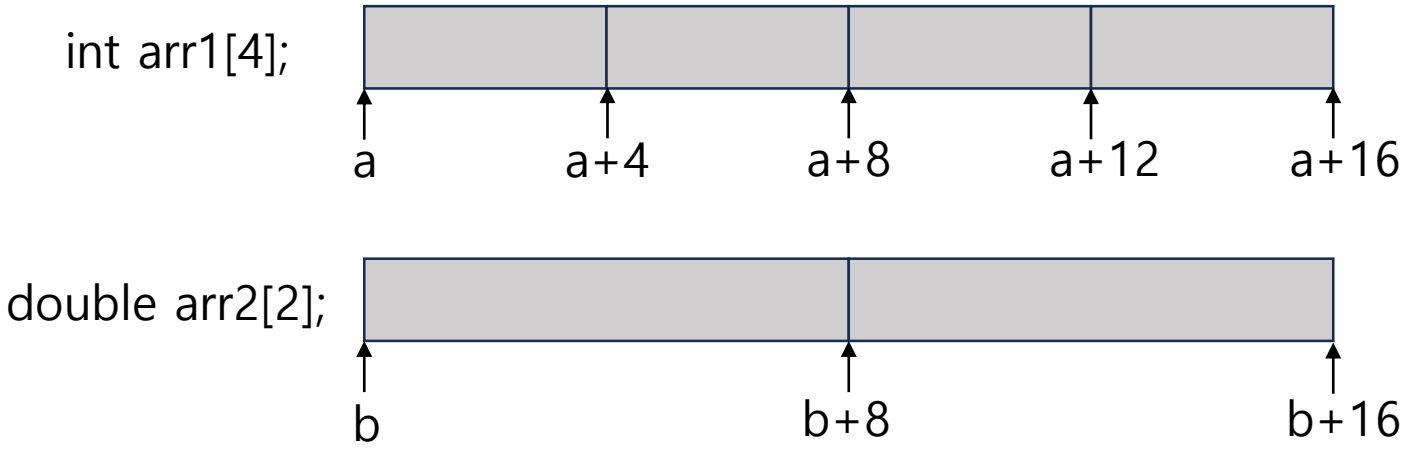
# MEMORY ALIGNMENT

- 특정 자료형의 크기가 K 바이트일 때, 해당 데이터의 시작 주소는 반드시 K의 배수여야 함

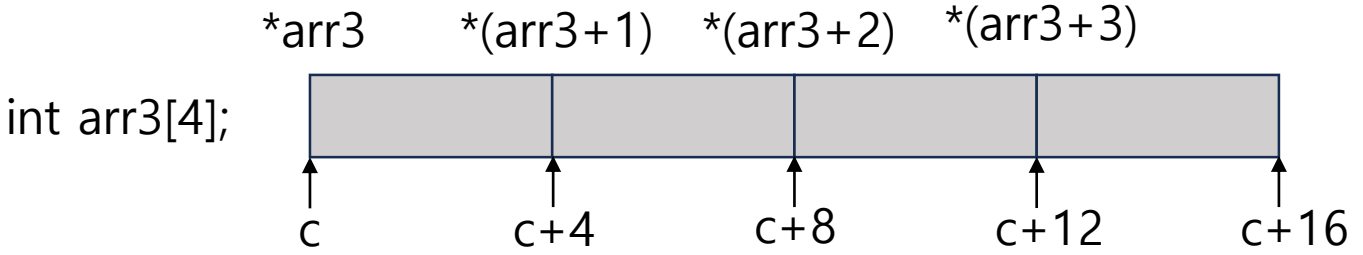
| 크기 | 자료형 | 주소 경계 |
|---|---|---|
| 1 | char | 제한 없음 |
| 2 | short | 2의 배수 |
| 4 | int, float, long, * | 4의 배수 |
| 8 | double, long long | 8의 배수 |

# HOMOGENEOUS ALIGNMENT

- Type arr[N]; -> 자료형이 Type이고 길이가 N인 배열 arr
- 동일한 타입 Type이 연속적인 메모리 영역에 할당됨




- 식별자 arr은 배열의 시작점을 가리키는 포인터로 사용될 수 있다.-> type of arr : Type*



# EXAMPLE: ARRAY

```
rtems_task Init(rtems_task_argument ignored)
{
  (void) ignored;
  TEST_BEGIN();
  volatile int arr1[2] = {10, 20};
  volatile char arr2[3] = {'A', 'B', 'C'};
  volatile double arr3[2] = {2.5, 4.5};
  TEST_END();
  exit(0);
}
```

*array-example/main.c*

```
(gdb) p &arr1
  $1 = (volatile int (*)[2]) 0x2a078
(gdb) p &arr2
  $2 = (volatile char (*)[3]) 0x2a070
(gdb) p &arr3
  $3 = (volatile double (*)[2]) 0x2a088
(gdb) p sizeof(arr1)
  $4 = 8
(gdb) p sizeof(arr2)
  $5 = 3
(gdb) p sizeof(arr3)
  $6 = 16
(gdb) x/8bx &arr1
  0x2a078:      0x00    0x00    0x00    0x0a    0x00    0x00    0x00    0x14
```

메모리 정렬 확인

Big-endian확인

# EXAMPLE: POINTER

```
int add(int x, int y) { return x + y; }
rtems_task Init(rtems_task_argument ignored)
{
  (void) ignored;
  TEST_BEGIN();
  volatile int (*f)(int, int) = add;
  volatile int A1[3];
  volatile int *A2[3];
  volatile char (*A3)[3];
  volatile char *A4[3][5];
  volatile double (*A5)[3][5];
  volatile double (*A6[3])[5];
  TEST_END();
  exit(0);
}
```

*pointer-example/main.c*

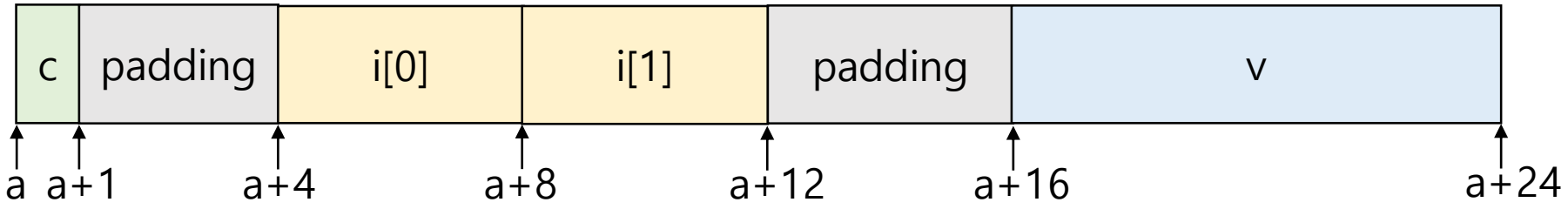
```
(gdb) p sizeof(add)
  $1 = 1
(gdb) p sizeof(f)
  $2 = 4
(gdb) p sizeof(A1)
  $3 = 12
(gdb) p sizeof(A2)
  $4 = 12
(gdb) p sizeof(A3)
  $5 = 4
(gdb) p sizeof(A4)
  $6 = 60
(gdb) p sizeof(A5)
  $7 = 4
(gdb) p sizeof(A6)
  $8 = 12
```

32-bit SPARC V8 아키텍처 기반
시스템에서 포인터가 4 byte 임을확인

# HETEROGENOUS ALIGNMENT: STRUCT

- 컴파일러는 구조체 내에 선언된 필드의 순서를 임의로 변경하지 않고 유지함.
  - 개발자의 의도된 데이터 배치를 존중하지만, 이로 인해 비효율적인 패딩이 발생할 수 있음.
- 구조체 내부의 각 필드는 해당 자료형의 정렬 규칙을 준수해야 함.
  - 정렬 조건을 충족하기 위해 필드 사이에 빈 공간을 삽입.
- 구조체 전체 주소 및 크기는 내부 멤버 중 가장 큰 정렬 단위를 기준으로 결정됨.

```
struct S1 {
  char c;
  int i[2];
  double v;
 };
```



# EXAMPLE: STRUCT

```
struct S1
{ char c; int i[2]; double v; };
rtems_task Init(rtems_task_argument ignored)
{
  (void) ignored;
  TEST_BEGIN();
  volatile struct S1 s;
  s.c='a';
  s.i[0]=10;
  s.i[1]=20;
  s.v=3.14;
  TEST_END();
  exit(0);
}
```

*struct-example/main.c*

```
(gdb) p sizeof(s)
  $1 = 24
(gdb) p &s
  $2 = (volatile struct S1 *) 0x26670
(gdb) p &s.c
  $3 = 0x26670 "a"
(gdb) p &s.i[0]
  $4 = (int *) 0x26674
(gdb) p &s.i[1]
  $5 = (int *) 0x26678
(gdb) p &s.v
  $6 = (double *) 0x26680
(gdb) x/24bx &s
  0x26670:      0x61    0x00    0x00    0x00    0x00    0x00    0x00    0x0a
  0x26678:      0x00    0x00    0x00    0x14    0x00    0x00    0x00    0x00
  0x26680:      0x40    0x09    0x1e    0xb8    0x51    0xeb    0x85    0x1f
```

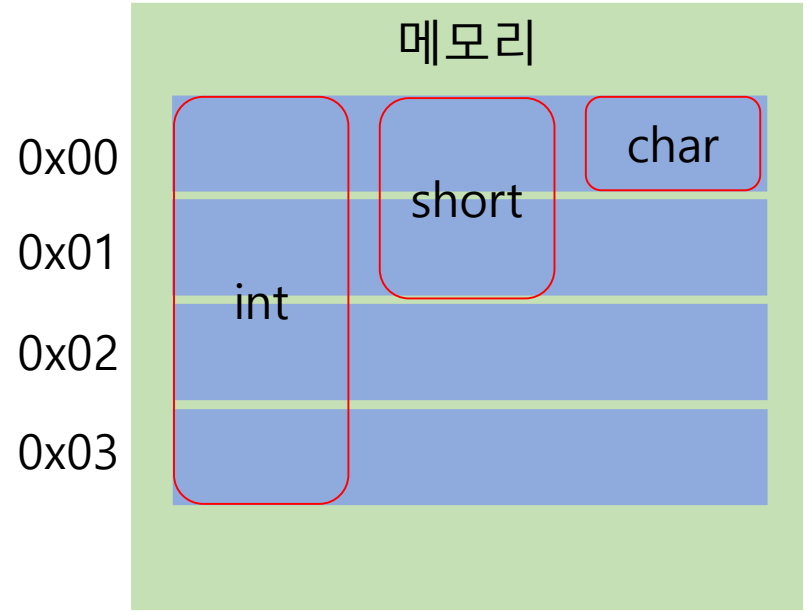
메모리 정렬 확인

Padding 확인

# HETEROGENOUS ALIGNMENT: UNION

- 메모리 공간 구성 및 매핑
  - 모든 멤버가 물리적으로 동일한 시작 지점을 가짐.
  - 메모리 오버레이: 메모리를 옆으로 나열하는 구조체와 달리, 동일한 주소 공간을 두고 어떤 자료형으로 읽을지만 결정하여 재사용하는 방식
  - 배타적 자원 사용: 동일한 물리 공간을 점유하므로, 특정 시점에는 하나의 멤버 데이터만 유효성을 가짐.
- 규격 및 정렬
  - 공용체 크기: 내부 멤버 중 가장 큰 자료형의 크기를 기준으로 할당됨.
  - 정렬 규칙: 멤버 중 가장 엄격한 정렬 제약을 가진 자료형을 기준점으로 채택함.
    EX) char와 double이 포함된 공용체는 double의 규칙에 따라 8의 배수 주소에 배치됨.

```
union MyUnion {
    char a
    short b
    int c
};
```



# EXAMPLE: UNION

```
union MyUnion {
    int i;
    char c[4];
};
rtems_task Init(rtems_task_argument
ignored)
{
(void) ignored;
TEST_BEGIN();
volatile union MyUnion u;
u.c[0] = 'A';
u.c[1] = 'B';
u.c[2] = 'C';
u.c[3] = 'D';
u.i = 0x12345678;
TEST_END();
exit(0);
}
```

*union-example/main.c*

```
(gdb) p sizeof(u)
  $1 = 4
(gdb) p sizeof(u.i)
  $2 = 4
(gdb) n
  20      TEST_BEGIN();
(gdb) n
  22      u.c[0] = 'A';
(gdb) n
  23      u.c[1] = 'B';
(gdb) n
  24      u.c[2] = 'C';
(gdb) n
  25      u.c[3] = 'D';
(gdb) n
  26      u.i = 0x12345678;
(gdb) p/x u.c
  $3 = {0x41, 0x42, 0x43, 0x44}
(gdb) p &u.i
  $4 = (int *) 0x26684
(gdb) p &u.c[0]
  $5 = 0x26684 "ABCD"
(gdb) x/4xb &u
  0x26684:      0x41    0x42    0x43    0x44
(gdb) n
  27      TEST_END();
(gdb) x/4xb &u
  0x26684:      0x12    0x34    0x56    0x78
```

공용체의 전체 크기가 가장 큰 멤버(int)의 크기인 4바이트로 결정됨을 확인

시작주소 일치확인

덮어쓰기 확인

# EXERCISE 1: MEMORY ALIGNMENT

- **1. 다음 빈칸에 들어갈 값을 작성하시오.**

```
struct S1 {
    char   a; double b; int    c;
};
struct S2 {
    double a;  Int    b; char   c;
};
union MyUnion {
    char   a; double b; int    c;
};
rtems_task Init(rtems_task_argument ignored)
{
(void) ignored;
TEST_BEGIN();
volatile struct S1 s1;
volatile struct S2 s2;
volatile union MyUnion u1;
volatile void* ptr = &s1;
printf("S1: %d\n", sizeof(s1));
printf("S2: %d\n", sizeof(s2));
printf("MyUnion: %d\n", sizeof(u1));
printf("Pointer: %d\n", sizeof(ptr));
TEST_END();
exit(0);
}
```

S1: _____

S2: _____

MyUnion: _____

Pointer: _____

# EXERCISE 1: MEMORY ALIGNMENT

- **문제 해설**
  - 정답: S1: 24

    S2: 16

    MyUnion: 8

    Pointer: 4
  - S1: 1(char) + 7(padding) + 8(double) + 4(int) + 4(padding) = 24
  - S2: 8(double) + 4(int) + 1(char) + 3(padding) = 16$
  - MyUnion: 멤버 중 가장 큰 자료형인 double을 기준으로 할당.
  - Pointer: 32-bit SPARC V8 아키텍처 기반 시스템에서 포인터가 4 byte로 고정됨.