

컴퓨터 구조 과제 3

컴퓨터 구조 3분반
소프트웨어학과 32170578 김산

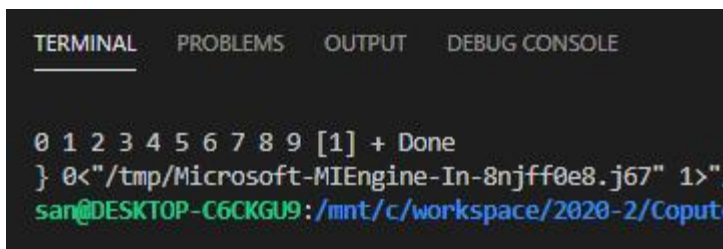
A. Write and test a C version that references the array elements with pointers, rather than using array indexing.

<C 코드>

```
#include<stdio.h>
#include<stdlib.h>
/*Bubble sort : Pointer version*/
void bubble_p(long* data, long count) {
    long *i, *last;
    for (last = data+count-1; last > data; last--) {
        for (i = data; i < last; i++) {
            if (*(i+1) < *i) {
                /* swap adjacent elements */
                long t =*(i+1);
                *(i+1) =*i;
                *i = t;
            }
        }
    }
}
int main(){
    long data[10] = {1,7,3,0,4,2,5,9,8,6};
    /*Excute bubble sort*/
    bubble_p(data, 10);
    /*Sort result*/
    for (int i =0; i <10; i++)
    {
        printf("%ld ", *(data+i));
    }
    return 0;
}
```

<실행 결과>

{1,7,3,0,4,2,5,9,8,6} 행렬 Bubble sort 수행



```
TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE

0 1 2 3 4 5 6 7 8 9 [1] + Done
} 0<"/tmp/Microsoft-MIEngine-In-8njff0e8.j67" 1>"
san@DESKTOP-C6CKGU9:/mnt/c/workspace/2020-2/Comput
```

B. Write and test a Y86-64 program consisting of the function and test code.

bubble_p함수 X86-64 -> Y86-64 풀이과정

X86-64	Y86-64	
bubble_p: .LFB41:	bubble_p: .LFB41:	rdi와 rsi의 값은 main함수에서 함수 호출 전 값을 삽입합니다.
leaq -8(%rdi,%rsi,8), %rsi movq %rdi, %r8 cmpq %rsi, %rdi jnb .L12	rrmovq %rdi, %r8 rrmovq %rdi, %r9 subq %rsi, %rdi jge .L12	cmpl 명령어의 경우 sub 연산을 하지만 값을 저장하지 않으므로 다른 레지스터(%r9)에 값을 저장하여 레지스터의 값을 수정하지 않으면서 sub 연산을 할 수 있습니다.
.L7: movq %rdi, %rax	.L7: rrmovq %rdi, %rax	
.L6: movq 8(%rax), %rdx movq (%rax), %rcx cmpq %rcx, %rdx jge .L5 movq %rcx, 8(%rax) movq %rdx, (%rax)	.L6: mrmovq 8(%rax), %rdx mrmovq (%rax), %rcx rrmovq %rdx, %r9 subq %rcx, %r9 jge .L5 rmmovq %rcx, 8(%rax) rmmovq %rdx, (%rax)	jnb는 작지않으면 조건이므로 같거나 크다는 조건과 같습니다. 따라서 jge로 바꿀수 있습니다.
.L5: addq \$8, %rax cmpq %rsi, %rax jb .L6 subq \$8, %rsi cmpq %r8, %rsi ja .L7 ret	.L5: irmovq \$8, %r9 addq %r9, %rax rrmovq %rax, %r9 subq %rsi, %r9 jl .L6 subq %r9, %rsi rrmovq %rsi, %r9 subq %r8, %r9 jg .L7 ret	Y86-64에서는 데이터가 어디로 이동하는지에 따라 ir, rm, mr, rrmovq의 명령어로 나뉩니다.
.L12: ret	.L12: ret	산술, 논리연산은 레지스터를 대상으로만 지원되므로 상수를 담은 레지스터를 사용합니다.
		jb -> jl ja -> jg

Y86-64코드

main:	
.align 20	
data:	
.quad	0x0000000000000001
.quad	0x0000000000000007
.quad	0x0000000000000003
.quad	0x0000000000000000
.quad	0x0000000000000004
.quad	0x0000000000000002
.quad	0x0000000000000005
.quad	0x0000000000000009
.quad	0x0000000000000008
.quad	0x0000000000000006
count:	
.quad	0x0000000000000010
main:	
irmovq	data, %rdi
irmovq	count, %rsi
call	bubble_p
ret	
bubble_p:	
rrmovq	%rdi, %r8
rrmovq	%rdi, %r9
subq	%rsi, %rdi
jge	.L12
.L7:	
rrmovq	%rdi, %rax
.L6:	
mrmovq	8(%rax), %rdx
mrmovq	(%rax), %rcx
rrmovq	%rdx, %r9
subq	%rcx, %r9
jge	.L5
rmmovq	%rcx, 8(%rax)
rmmovq	%rdx, (%rax)
.L5:	
irmovq	\$8, %r9
addq	%r9, %rax
rrmovq	%rax, %r9
subq	%rsi, %r9
jb	.L6
subq	%r9, %rsi
rrmovq	%rsi, %r9
subq	%r8, %r9
ja	.L7
ret	
.L12:	
ret	

Bonus(30%): 교재의 bubble sort 예제 코드를 insertion sort로 변경하여 풀이

A. Write and test a C version that references the array elements with pointers, rather than using array indexing.

<C 코드>

```
#include<stdio.h>
#include<stdlib.h>
/*Insertion_sort : Pointer version*/
void insertion_sort(long* data, long count) {
    long *i, *j, temp;
    for(i=data+1; i<data+count; i++){
        temp =*i;
        for(j=i-1; j>=data && *j>temp; j--){
            *(j+1) =*(j);
        }
        *(j+1) = temp;
    }
}
int main(){
    long data[10] = {1,7,3,0,4,2,5,9,8,6};
    /*Excute insertion_sort sort*/
    insertion_sort(data, 10);
    /*Sort result*/
    for (int i =0; i <10; i++)
    {
        printf("%ld ", *(data+i));
    }
    return 0;
}
```

<실행 결과>

{1,7,3,0,4,2,5,9,8,6} 행렬 Insertion sort 수행

B. Write and test a Y86-64 program consisting of the function and test code.

insertion_sort 함수 X86-64 -> Y86-64 풀이과정

X86-64	Y86-64	
insertion_sort: .LFB41:	insertion_sort: .LFB41:	cmpl 명령어의 경우 sub 연산을 하지만 값을 저장하지 않으므로 다른 레지스터(%r10)에 값을 저장하여 레지스터의 값을 수정하지 않으면서 sub 연산을 할 수 있습니다. jnb -> jge
leaq (%rdi,%rsi,8), %rax leaq 8(%rdi), %rdx cmpq %rsi, %rdi jnb .L1 subq %rdi, %rax movq %rdi, %rcx leaq -9(%rax), %r9	rrmovq %rdi, %r10 subq %rsi, %r10 jge .L1 subq %rdi, %rax rrmovq %rdi, %rcx mrmovq -9(%rax), %r9	

andq \$-8, %r9	irmovq \$-8, %r10	산술, 논리연산은 레지스터를 대상으로만 지원되므로 상수를 담은 레지스터(%r10)를 사용합니다. ja -> jg neg 명령어는 -1과 XOR연산 한 후 +1하여 구현할 수 있다.
addq %rdx, %r9	andq %r10, %r9	
.L5:	.L5:	
cmpq %rcx, %rdi	rrmovq %rdi, %r10	
leaq 8(%rcx), %r8	subq %rcx, %r10	
movq %rcx, %rax	rrmovq 8(%rcx), %r8	
movq 8(%rcx), %rsi	rrmovq %rcx, %rax	
ja .L3	rrmovq 8(%rcx), %rsi	
movq (%rcx), %rdx	jg .L3	
cmpq %rdx, %rsi	rrmovq (%rcx), %rdx	
jge .L3	rrmovq %rsi, %r10	
subq %rdi, %rcx	subq %rdx, %r10	
andq \$-8, %rcx	jge .L3	
negq %rcx	subq %rdi, %rcx	
leaq -16(%r8,%rcx), %rcx	irmovq \$-8, %r10	
jmp .L4	andq %r10, %rcx	
	irmovq \$-1, %r10	
	irmovq \$1, %r11	
	xorq %r10, %rcx	
	addq %r11, %rcx	
	rrmovq -16(%r8), %r10	
	rrmovq (%rcx), %r11	
	addq %r10, %r11	
	rmmovq %r11, (%rcx)	
	jmp .L4	
.L12:	.L12:	
movq (%rax), %rdx	rrmovq (%rax), %rdx	
cmpq %rsi, %rdx	rrmovq %rdx, %r10	
jle .L11	subq %rsi, %r10	
.L4:	.L4:	
movq %rdx, 8(%rax)	rrmovq %rdx, 8(%rax)	
subq \$8, %rax	irmovq \$8, %r10	
cmpq %rcx, %rax	subq %r10, %rax	
jne .L12	rrmovq %rax, %r10	
.L3:	.L3:	
cmpq %r8, %r9	rrmovq %r9, %r10	
movq %rsi, 8(%rcx)	subq %r8, %r10	
movq %r8, %rcx	rmmovq %rsi, 8(%rcx)	
jne .L5	rrmovq %r8, %rcx	
.L1:	.L1:	
ret	ret	
.L11:	.L11:	
movq %rax, %rcx	movq %rax, %rcx	
cmpq %r8, %r9	rrmovq %r9, %r10	
movq %rsi, 8(%rcx)	subq %r8, %r10	
movq %r8, %rcx	rmmovq %rsi, 8(%rcx)	
jne .L5	rrmovq %r8, %rcx	
jmp .L1	jne .L5	
	jmp .L1	

main:	
.align 20	
data:	
.quad	0x0000000000000001
.quad	0x0000000000000007
.quad	0x0000000000000003
.quad	0x0000000000000000
.quad	0x0000000000000004
.quad	0x0000000000000002
.quad	0x0000000000000005
.quad	0x0000000000000009
.quad	0x0000000000000008
.quad	0x0000000000000006
count:	
.quad	0x0000000000000010
main:	
irmovq	data, %rdi
irmovq	count, %rsi
call	bubble_p
ret	
bubble_p:	
.LFB41:	
rrmovq	%rdi, %r10
subq	%rsi, %r10
jge	.L1
subq	%rdi, %rax
rrmovq	%rdi, %rcx
mrmovq	-9(%rax), %r9
irmovq	\$-8, %r10
andq	%r10, %r9
addq	%rdx, %r9
.L5:	
rrmovq	%rdi, %r10
subq	%rcx, %r10
mrmovq	8(%rcx), %r8
rrmovq	%rcx, %rax
mrmovq	8(%rcx), %rsi
jb	.L3
mrmovq	(%rcx), %rdx
rrmovq	%rsi, %r10
subq	%rdx, %r10
jge	.L3
subq	%rdi, %rcx
irmovq	\$-8, %r10
andq	%r10, %rcx
irmovq	\$-1, %r10
irmovq	\$1, %r11
xorq	%r10, %rcx
addq	%r11, %rcx
mrmovq	-16(%r8), %r10
mrmovq	(%rcx), %r11
addq	%r10, %r11
rrmovq	%r11, (%rcx)
jmp	.L4
.L12:	
mrmovq	(%rax), %rdx

rrmovq	%rdx, %r10
subq	%rsi, %r10
jle	.L11
.L4:	
rmmovq	%rdx, 8(%rax)
irmovq	\$8, %r10
subq	%r10, %rax
rrmovq	%rax, %r10
sub	%r10, %rax
jne	.L12
.L3:	
rrmovq	%r9, %r10
subq	%r8, %r10
rmmovq	%rsi, 8(%rcx)
rrmovq	%r8, %rcx
jne	.L5
.L1:	
ret	
.L11:	
movq	%rax, %rcx
rrmovq	%r9, %r10
subq	%r8, %r10
rmmovq	%rsi, 8(%rcx)
rrmovq	%r8, %rcx
jne	.L5
jmp	.L1