REPORT

[어셈블리 실습 02]

과 목 : 시스템소프트웨어
담당교수 : 석문기 교수님
학 과 : 컴퓨터공학과
학 번 : 2021111971
이 름 : 이재혁

dongguk UNIVERSITY

1. 어셈블리어 코드

jump_to_middle_loop_sum.asm

```
section .text
global jump_to_middle_loop_sum
jump_to_middle_loop_sum:
    mov edx, 0 ; sum 0 으로 초기화
    mov eax, 0 ; i 0 으로 초기화
    jmp .loop_check ; 처음에 조건 검사 부분으로 점프
.loop_start:
    movsxd rcx, eax ; i 값을 64 비트로 부호 확장하여 rcx 에 저장
    add edx, dword[rdi+rcx*4] ; sum+=arr[i]
    add eax, 1        ; i++
    jmp .loop_check ; 조건 검사 부분으로 점프
.loop_check:
    cmp eax, esi    ; i < n 검사
    jl .loop_start ; i < n 이면 .loop_start 로 이동
    mov eax, edx    ; sum 에 저장된 값 eax 에 저장
    ret             ; 함수 반환
```

guarded_do_loop_sum.asm

```
section .text
global guarded_do_loop_sum
guarded_do_loop_sum:
; todo: 초기화
    mov edx, 0      ; sum = 0
    mov eax, 0      ; i = 0
; todo: 초기조건 검사
    cmp eax, esi   ; i < n
    jge .loop_end ; i >= n 이면.loop_end 로 이동해 반복 종료
.loop_start:
; todo: body
    movsxd rcx, eax ; i 64 비트 확장
    add edx, dword[rdi + rcx * 4] ; sum += arr[i]
    add eax, 1       ; i++
; todo: test
    cmp eax, esi    ; i < n
    jl .loop_start   ; i < n 이면 다시 반복
.loop_end:
    mov eax, edx ; 최종 합계를 eax 에 저장
    ret; 함수 반환 (eax 에 최종 합계가 저장됨)
```

2. 컴파일 후 실행결과

```
[2021111971@linuxserver1:~$ ./loop_sum_conversion_test
Sum using jump-to-middle loop: 15
Sum using guarded-do loop: 15
```

3. for_loop_ctest.s

for_loop_sum

```
for_loop_sum:
.LFB0:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        movq    %rdi, -24(%rbp) ; arr 의 포인터를 rdi 저장
        movl    %esi, -28(%rbp) ; n 의 값을 esi 에 저장
        movl    $0, -8(%rbp)    ; sum 0 으로 초기화
        movl    $0, -4(%rbp)    ; i 를 0 으로 초기화
        jmp     .L2             ; .L2 조건 검사 부분으로 점프
.L3:
        movl    -4(%rbp), %eax  ; i 를 eax 에 저장
        cltq                    ; eax 를 64 비트로 확장
        leaq    0(,%rax,4), %rdx; i*4 인덱스 위치를 저장
        movq    -24(%rbp), %rax ; 배열의 시작주소를 rax 에 저장
        addq    %rdx, %rax      ; arr[i]의 주소 계산
        movl    (%rax), %eax    ; arr[i]의 값 eax 에 저장
        addl    %eax, -8(%rbp)  ; sum += arr[i]
        addl    $1, -4(%rbp)    ; i++
.L2:
        movl    -4(%rbp), %eax  ; i 값 eax 에 저장
        cmpl    -28(%rbp), %eax ; i < n 비교
        jl      .L3             ; i < n 이면 .L3 로 반복
        movl    -8(%rbp), %eax  ; eax 에 sum 저장
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
```

초기화 이후, 조건 확인 -> jump to middle 변환

4. dot_product.asm

```
section .text
global dot_product
dot_product:
    ; 초기화
    mov eax, 0                      ; sum = 0
    mov ecx, 0                      ; i = 0
.loop_start:
    ; i < n 검사
    cmp ecx, edx                    ; i < n 비교
    jge .loop_end                   ; i >= n 이면 루프 종료
    ; vec1[i] * vec2[i] 계산
    movsxd rcx, ecx                 ; i를 64비트 rcx로 확장
    mov ebx, dword [rdi + rcx * 4]  ; vec1[i] 값을 ebx에 저장
    imul ebx, dword [rsi + rcx * 4] ; ebx와 vec2[i]곱 ebx에 저장
    ; sum += vec1[i] * vec2[i]
    add eax, ebx                    ; eax에 곱셈 결과를 더함
    ; i++
    inc ecx                         ; 인덱스 i를 증가
    ; 루프 반복
    jmp .loop_start
.loop_end:
    ; 결과 반환 (eax에 최종 내적 값 저장됨)
    ret
```

5. dot_product_test.c 테스트

```
2021111971@linuxserver1:~$ nano dot_product.asm
2021111971@linuxserver1:~$ nasm -f elf64 dot_product.asm -o dot_product.o
2021111971@linuxserver1:~$ gcc dot_product_test.c dot_product.o -o dot_product_t
est
2021111971@linuxserver1:~$ ./dot_product_test
Dot product of vec1 and vec2: 130
```