**REPORT**

**[어셈블리 실습 05]**

원, 폰트, 동전이(가) 표시된 사진

자동 생성된 설명

**과 목 : 시스템소프트웨어**

**담당교수 : 석문기 교수님**

**학 과 : 컴퓨터공학과**

**학 번 : 2021111971**

**이 름 : 이재혁**

텍스트, 클립아트이(가) 표시된 사진

자동 생성된 설명

**1. Scalar Operation**

|  |  |
| --- | --- |
| scalar\_vector\_mul\_sum.asm | |
| section .data  zero dd 0.0 ; float 0.0  section .text  global scalar\_vector\_mul\_sum  ; void scalar\_vector\_mul\_sum(float \*a, float \*b, float \*result, float \*sum)  scalar\_vector\_mul\_sum:  ; 레지스터 매핑  ; a -> rdi, b -> rsi, result -> rdx, sum -> rcx  ; 초기화  xor r8, r8 ; r8 = 0 (루프 카운터 i)  movss xmm3, dword [rel zero] ; xmm3 = 0.0 (합계 초기화)  .loop\_start:  ; i가 4보다 크거나 같은지 확인  **cmp r8, 4; TODO: if (i >= 4) break;**  jge .loop\_end  ; a[i]를 xmm0에 로드  mov rax, r8 ; rax = i  **shl rax, 2 ; TODO: rax = i \* 4 (float 크기 계산), i<<2로**  **movss xmm0, dword[rdi+rax] ; TODO: xmm0 = a[i]**  ; b[i]를 xmm1에 로드  movss xmm1, dword[rsi + rax] ; xmm1 = b[i]  ; xmm0 = xmm0 \* xmm1  mulss xmm0, xmm1 ; result[i]에 저장  **movss dword[rdx+rax], xmm0; TODO: result[i] = xmm0**  ; 합계에 추가  addss xmm3, xmm0  inc r8 ; i++  ; 루프 반복  jmp .loop\_start  .loop\_end: ; 합계를 sum에 저장  **movss dword[rcx], xmm3; TODO: \*sum = xmm3**  ; 함수 종료  ret | rax = r8(index) \* 4byte  -> 계산할 값의 위치입니다.  dword[rdi + rax]  -> a[i]  dword[rsi + rax]  -> b[i]  C언어 표현  for(int i = 0; i < 4; i++) {  result[i] = a[i] \* b[i];  sum += result[i];  } |

**실행결과**

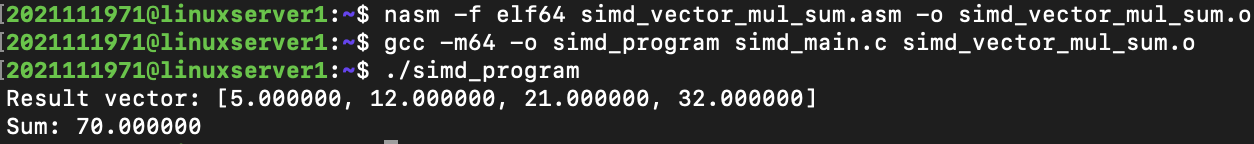
텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

**2. Vector Operation**

|  |  |
| --- | --- |
| scalar\_vector\_mul\_sum.asm | |
| section .text  global simd\_vector\_mul\_sum  ; void simd\_vector\_mul\_sum(float \*a, float \*b, float \*result, float \*sum)  simd\_vector\_mul\_sum:  ; 레지스터 매핑  ; a -> rdi, b -> rsi, result -> rdx, sum -> rcx  ; 1. 벡터 데이터 로드  vmovaps xmm0, [rdi] ; rdi(a)에서 4개의 float 값을 xmm0으로 로드  vmovaps xmm1, [rsi] ; rsi(b)에서 4개의 float 값을 xmm1으로 로드  ; 2. 원소별 곱셈  vmulps xmm2, xmm0, xmm1 ; xmm2 = xmm0 \* xmm1  ; 3. 곱셈 결과를 result 배열에 저장  **vmovaps [rdx], xmm2 ; TODO: xmm2를 rdx(result) 메모리에 저장**  ; 4. 벡터 합계 계산 (수평 덧셈)  vhaddps xmm2, xmm2, xmm2 ; xmm2의 4개 값을 수평 덧셈으로 합산  **vhaddps xmm2, xmm2, xmm2 ; TODO: 두 번째 수평 덧셈으로 최종 합계 계산**  ; 5. 합계를 sum 변수에 저장  **vmovss dword[rcx], xmm2; TODO: rcx(sum) 위치에 첫 번째 원소 저장**  ; 6. 함수 종료  ret | [rdi]  -> a의 값 들의 벡터입니다.  1.0, 2.0, 3.0, 4.0  [rsi]  -> b의 값 들의 벡터입니다.  5.0, 6.0, 7.0, 8.0  xmm2  -> 원소 별 곱을 저장합니다.  5.0, 12.0, 21.0, 32.0  vmoss [rdx], xmm2  rdx가 result의 주소이므로  xmm2 벡터의 값이 result에 저장됩니다.  vhaddps 2번의 연산으로 xmm2의 첫 번째 원소에 모든 원소의 합이 저장되었습니다.  나머지 원소는 사용하지 않는 값입니다.  sum에 값을 저장합니다. |

**실행결과**



**SIMD, 스칼라연산 비교**SIMD방식은 다수의 값을 한번에 벡터로 저장하기 때문에, 적은 주소 접근, 합 연산에서 병렬처리로 인해 스칼라 연산보다 효율적입니다.

**3. 비교 명령어 (ucomiss, ucomisd)**

|  |
| --- |
| compare.asm |
| section .text  global compare\_floats  global compare\_doubles  ; int compare\_floats(float a, float b)  compare\_floats:  ; 매개변수: a -> xmm0, b -> xmm1  **ucomiss xmm0, xmm1; TODO: float a와 float b 비교**  jp .nan\_case ; NaN일 경우 처리  jb .less ; a < b: CF = 1  ja .greater ; a > b: CF = 0, ZF = 0  je .equal ; a == b: ZF = 1  .less:  **mov eax, -1 ; 반환 값: -1 (a < b)**  ret  .greater:  **mov eax, 1 ; TODO: 반환 값: 1 (a > b)**  ret  .equal:  **mov eax, 0 ; TODO: 반환 값: 0 (a == b)**  ret  .nan\_case:  **mov eax, -2 ; NaN 경우: 반환 값 -2**  ret  ; int compare\_doubles(double a, double b)  compare\_doubles:  ; 매개변수: a -> xmm0, b -> xmm1  **ucomisd xmm0, xmm1 ; TODO: double a와 double b 비교**  jp .nan\_case\_d ; NaN일 경우 처리  jb .less\_d ; a < b: CF = 1  ja .greater\_d ; a > b: CF = 0, ZF = 0  je .equal\_d ; a == b: ZF = 1  .less\_d:  **mov eax, -1; TODO: 반환 값: -1 (a < b)**  ret  .greater\_d:  **mov eax, 1 ; TODO: 반환 값: 1 (a > b)**  ret  .equal\_d:  **mov eax, 0 ; TODO: 반환 값: 0 (a == b)**  ret  .nan\_case\_d:  **mov eax, -2; TODO: NaN 경우: 반환 값 -2**  ret |

**실행결과**

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

float\_result 전달 인자 값

a : 5.5, b : 3.3 -> return 1

double\_result 전달 인자 값

a : 3.3, b : 3.3 -> return 0