과제 <PA1 : MIPS Assembly to Machine Instruction Translator >

과제 목적 : MIPS 어셈블리 코드를 MIPS 기계 명령어로 변환하는 것입니다.

<분석 전략>

1. Register 및 숫자 입력 부분을 10진법 숫자로 변환

각 명령어 뒤에는 숫자나 t0, s1 같은 레지스터가 올 수 있습니다. 레지스터값은 0부터 31까지로 이루어지며, 각 레지스터마다 주소 값이 고정 되어있습니다. 따라서 해당 레지스터 값을 10진수로 변환하여 내보내거나, 숫자가 왔을 때는 숫자를 그대로 return 하는 함수를 `register_translate`로 정의하였습니다. 입력 parameter는 토큰 단위로 받아온 값을 넣을 수 있습니다.

1) Register 입력 부분

각 시작 부분 알파벳을 비교하여 케이스를 나눈 뒤에, 다음 숫자에 따라 순차적으로 값이 증가하는 (예를 들어 t0~ t7 이 8~15 로 구성됨) 경우가 많으므로 알파벳 뒤에 오는 숫자를 atoi 로 변환해 값을 맞춰 return 하였습니다. (하지만 전체 케이스 중 이에 해당되지 않는 케이스가 더 많아서 각 register 이름으로 구성된 array와 각 register 에 해당하는 숫자 array를 따로 만들어 각 index를 일치시키는 방식으로 구현하는 것이 좀 더 효율적 일 것 같습니다.(예를 들어 [t0,t1,t2 ..] [8,9,..] 같은 식으로 index를 일치시켜 해당 index에 해당하는 숫자를 return 하는 방식)) 해당하지 않은 케이스들, 예를 들어 s로 동일하게 시작하지만 sp와 s + 숫자로 구성된 register같은 경우들은 따로 조건을 걸어주어 해결했습니다.

2) 숫자 입력 부분 (16진수)

Register로 입력되는 토큰들은 모두 알파벳으로 시작하므로, 알파벳으로 시작되지 않은 토큰들은 숫자로 간주합니다. 모두 숫자로 받는다고 생각했을 때, 고려해야할 부분은 10진수와 16진수를 구별해야한다는 것입니다. 16진수는 0x로 시작하므로 토큰의 첫번째 문자가 0이라면 16진수로 생각하고 strtol함수를 통해 변환해 내보냅니다. 만약 0으로 시작하는데 16진수가 아니라 10진수라면, 그 경우는 016 같은 식으로 10진수가 들어오지 않는 이상 0일 수 밖에 없는데 0은 16진수와 10진수 동일하게 표현하므로 문제가 없을 것이라 판단했습니다.

3) 숫자 입력 부분 (10진수)

1), 2), 4) 번에 해당하지 않는 모든 경우는 10진수 이므로 string을 그대로 atoi로 변환해서 내보냅니다.

4) 숫자 입력 부분 (음수)

음수는 10진수와 16진수의 경우라면 무조건 '-' 로 시작하므로 -로 시작하는 문자열이라면 strtok로 -을 제외하고 2) 3)번과 동일한 순서를 진행 한 뒤 -를 다시 붙여 return 했습니다.

2. 10진법 숫자를 2진법으로 변환

Rtype 과 Itype 변수들의 bit수가 다르므로 이에 해당하는 숫자만큼의 bit수를 가진 2진법으로 변환시켜야 합니다. 때문에 len변수를 추가시켜 딱 맞아 떨어지게 2진법으로 변환시켰습니다. `tenConvertTwo` : Len만큼 먼저 문자열 세팅 해 준 뒤 보통 2진법 계산처럼 2로 계속 나눠

가면서 나머지를 뒤에서 채워주는 형태로 변환시켰습니다. 음수인 경우를 구별하기 위해 minus parameter를 추가시켜 minus가 -1인 경우 음수로 간주하고 변환 했습니다. 음수로 변환하는 과정은 - 를 제외한 값을 먼저 2진법으로 변환한 뒤, 2의 보수를 구하는 방식처럼 변환시켰습니다. 이때 마지막에 1을 더해 주는 연산은 작은 자리 수부터 차례대로 0을 찾은 뒤 1로 변환하고 그 아래 자리숫자들은 모두 0으로 변환했습니다. (1011 -> 1010 -> 1000 ->(0을 찾음) 1100)

3. Instruction type 구분

과제에서 구현해야하는 instruction type은 총 2가지로, R type과 I type입니다. 이 함수들은 각 변수들을 2진법으로 변환 뒤 32의 길이만큼 동적 할당한 문자열에 변환한 문자열을 합쳐 반환합니다.

1) R-type은 opcode, rs, rt, rd, shamt, funct 로 구성되어있으며, 해당 옵션들을 합치는 함수는 `R_type` 입니다. 구성요소들은 Int 값으로 받아 이를 2 에서 작성했던 2진법 코드를 통해 2진법으로 변환한 뒤 순서에 맞춰 합쳐 준 다음 반환했습니다.
2) I-type 은 opcode, rs, rt, imm 로 구성되어있으며, 해당 옵션들을 합치는 함수는 `I_type` 입니다. 1)과 마찬가지로 int값으로 받아 2 에서 작성했던 2진법 코드를 통해 2진법으로 변환 한 뒤 순서에 맞춰 합쳐 준 다음 반환했습니다.

4. 합친 2진법 숫자들을 10진법으로 변환 뒤 출력

`translate` 함수는 `parse_command` 에서 assembly를 공백 단위로 자른 token들을 가지고 mips 기계 명령어로 변환을 위한 10진수로 변환해서 return 합니다. token[0]를 가지고 R_type instruction인지, I_type instruction인지 확인하고, 각 instruction에 따라 opcode나 funct를 직접 입력, 또한 tokens들 중 피연산자들은 `register_translate`로 10진수로 변환 한 뒤 함수에 넣어주어 해당 assembly를 2진법으로 변환해서 넣어줍니다. 이후 main 에서 10진수로 값을 받아오기 때문에 2진수를 strtol을 통해 10진수로 변환 뒤 return 합니다.

5. Lesson learned

- 문제를 처음 접했을 때 어떠한 구조로 코드를 작성 해야하는지에 대해 설계하는 습관이 들게 되었습니다. 처음에 구조화하는 과정에서 수업에서 변환하는 과정을 공부했던 과정을 바탕으로 각 명령어를 2진법으로 변환, 그 다음 이어진 문자열을 16진법으로 변환하는 과정을 그대로 함수에 적용했습니다.
- Strtol 함수에 대해서 알게 되었습니다. Strtol 함수란 특정 진법으로 쓰인 문자열을 10진법으로 변환해주는 함수로서, main함수에서 int형태로 변환된 값을 받기 때문에 이를 strtol함수를 통해 return 해 주었습니다.
- 헤맸던 부분은 문자열을 합치는 과정에서 자꾸 **corrupted size vs. prev_size Aborted** 오류가 발생한다는 것이었습니다. 이는 동적 할당 시 배정한 size보다 넘게 값을 할당했을 경우 나타나는 오류로, 합치는 과정 속에서 이미 동적 할당 해놓은 문자열에다가 문자열을 더 덧붙여 오류가 계속 발생하고 있었습니다. 이를 해결하기 위해서 반환하는 문자열 크기만큼 새로 할당해 이에 대해서 붙이는 방식으로 해결했습니다. 수업시간에 배웠던 dynamic allocate하는 과정 속에서 heap에 어떻게 구조가 쌓이는지 이해하는 과정 속에 해결할 수 있었던 것 같습니다.