이번 과제는 instruction 과 register 값, 또는 constant/adress 값의 정보가 16진수로 입력되면, 그 instruction의 역할을 수행하여 연산을 수행하여, 올바른 register값에 맞는 값이 들어가고, memory값에 lw, sw등이 수행되여 제대로 된 연산을 수행하도록 하는 function들을 구현하는 과제였습니다. 구현해야 할 function 으로는 먼저 1. process_instruction() function. instruction이 R format인지 I format인지를 구별하고, 올바른 연산을 수행하도록 하는 함수. 2. load program() function. memory array에 16진수들을 저장하는 함수. 이때 마지막 배열 자리에는 0xffffffff가 저장되도록 하여, 프로그램의 끝을 구분할 수 있게 해야 합니다. 3. run_program() function. memory에 저장된 16진수를 차례대로 받아 instruction 함수에 매개변수로 받아 instruction 함수를 작동시키고, pc값을 이동시키는 함수 로 3가지가 있습니다.

먼저 process_instruction() function은 MIPS 연산을 수행합니다. 먼저 입력받은 16진수의 32bits중, 앞의 6bits만이 R-format/I-format을 구분할 수 있는 opcode입니다. 그래서 26bits를 shift시켜 form 변수에 저장한 후, format을 구분했습니다. 그 후 format에 따라 구성 형식이 다르므로, 먼저 R-format에 대해 rs, rt, rd, shamt 변수에 올바른 값을 and 연산을 통해 저장한 후, General R-format인 경우 type 변수에 26씩 왼쪽으로 shift한 후 오른쪽으로 다시 shift하여 function 코드를 저장하여, fucntion 코드에 맞는 instruction의 연산을 registers array의 rs, rt, rd 값을 index로 하여 계산을 하는 코드를 구현했습니다. shift R-format인 경우 rd, rt, shamt 변수를 사용하도록 했습니다. 이때 'sra' instruction은 rt값이 음수인 경우 shift한 후, srl, sll의 경우 0으로 채우는 부분을 숫자를 모두 1로 채워야 하기 때문에 rt의 첫 번째 부호 비트가 1인지를 확인한 후, 1이면 shamt만큼 이동한 부분의 앞을 1로 채우는 기능을 추가했습니다. I-format인경우 function코드가 없고 opcode로 instruction을 구분할 수 있으므로, form변수의 값에 따라 다른 역할을 하도록 하는 기능을 넣었습니다. 이때 constant 값이 음수일 경우가 있기 때문에 음수인 경우, 양수인 경우를 나누어 연산하도록 했습니다. 그 중 lw의 경우 4칸의 memory에 8bits씩 나누어져 있는 값을 32bits로 다시 합쳐야 하므로, shift 연산과 | 연산을 통해 32bits를 다시 만든 후 registers[rt] 자리에 넣습니다. sw의 경우 register에 있는 32bits를 8bits씩 나눠서 memory에 저장합니다. 이 과정 역시 shift연산을 이용하며, &연산을 사용합니다. 이때 andi와 ori는 const값만을 가지고 연산하므로 &연산을 통해 const값만으로 연산할 수 있도록 했습니다. 그리고 constant값 외의 address값을 이용하는 J-format의 경우, 먼저 &연산을 이용해 address 값을 구했습니다. j instruction은 address값의 맨 앞 4bits를 pc값의 맨 앞 4bits로 바꾸고, 왼쪽으로 2번 shift해야 한다는 특징을 가지고 있습니다. jal instruction은 pc의 값의 변화에 있어서는 j instruction과 하는 일이 같지만, 현재 pc값을 ra register에 저장한다는 역할을 수행해야 하므로, add변수에 저장된 address값을 왼쪽으로 shift한 후, pc값의 맨 앞 4bits를 shitf연산을 통해 구한 후 이들을 | 연산하여 올바른 pc값을 구하도록 했습

니다. 이 function에서 제일 처음에 해야 할 것은 현재 pc의 값이 0xffffffff을 가르키고 있다면 프로그램을 종료시켜야 하므로, 매개변수로 받는 instr 값이 0xffffffff이라면 0을 return하도록 했습니다.

두 번째 load program() function은 16진수들을 파일로부터 fopen과 fgets를 이용해 읽어온 후, strtoimax를 이용해서 파일의 16진수 부분만을 읽어옵니다. 이후 memory array의 4칸에 16진수로 표현된 MIPS 연산과 register, constant/adress값을 32bits를 8bits씩 각각 나누어 저장하는 기능을 합니다. 이때 & 연산을 통해 원하는 부분을 bit masking한 후 shift시켜 8bits만을 array의 한 칸에 저장합니다. 모든 파일의 값을 다 읽을 때 까지 while문을 이용해 반복시켜 memory 칸을 하나씩 증가시켜 저장하는 기능을 수행하게 했고, 파일을 다 읽은 후에는 마지막 8bits를 저장한 다음 4칸에 각 0xff값을 저장하여 0xffffffff pc가 파일이 끝나는 부분을 알 수 있도록 했습니다.

마지막 run_program() function의 가장 큰 역할은, process_instruction() function을 작동시킨다는 점입니다. memory array의 각 칸에 있던 8bits들을 왼쪽으로 shift한 후 32bits를 만들어 inst 변수에 넣은 후 process_instruction()에 매개변수로 전달합니다. process_instruction() function을 작동시켜 얻은 결과값이 result 변수에 저장되는데, process_instruction() function은 pc값이 0xffffffff을 만나게 되면 0을 return하도록 했기 때문에, result값이 0이 되면 함수를 종료합니다. 동시에 pc값을 4만큼 증가시키는 역할을 합니다.

이번 과제를 하면서 중간고사 시험 범위 중 제가 부족했던 부분을 다시 공부할 수 있었습니다. 특히 j-instruction 부분에서 저의 이해도가 많이 떨어졌었다는 것도 깨닫게 되었습니다. bitwise 연산과 masking을 자주 활용하면서 bitwise 연산이 얼마나 효율적인지를 다시 한 번 알 수 있었습니다.