How you process the MIPS instructions

1. opcode, funct, rd, rt, rs, shamt, imm, address, sign 변수
   우선 R포맷은 opcode(6bits) rs(5bits) rt(5bits) rd(5bits) shamt(5bits) funct(6bits) 인데 각각 모두 뽑아내야 하기 때문에 opcode는 26만큼, rs는 21만큼, rt는 16만큼 rd는 11만큼 shamt는 6만큼 오른쪽으로 시프트(<<연산)를 하고 &연산을 해주었습니다. 뽑아내야 한다는 의미를 다음과 같이 설명하겠습니다. 예를 들어, 0x02952024(# and a0 s4 s5) 를 입력한다면 이를 컴퓨터는 00000010100101010010000000100100으로 인식을 할 것이고, rt에 해당하는 레지스터를 알기 위해 오른쪽으로 16만큼 연산하면 00000000000000000000010100101010이고 0b00000000000011111와 &연산을 하면 맨 오른쪽 5bit 즉 rt에 해당하는 부분만 남기 때문에 00000000000000000000000000010101이 되고 이는 21이며 $s5임을 알 수 있습니다. 그래서 각 해당하는 레지스터를 알기 위해 opcode, funct, rd, rt, rs, shamt 부분을 맞게 변수로 두고 처리를 먼저 해주었습니다. I포맷은 opcode(6bits) rs(5bits) rt(5bits) address/constant(16bits) 이고, address/constant부분만 r포맷과 다르므로 이를 imm이라는 변수로 두고 16비트이므로 00000000000000001111111111111111 과 &연산을 해주었습니다. address라는 변수는 J포맷을 위한 것으로 26bit인 address를 위해 알맞게 &연산을 해주었으며 sign이라는 변수는 sra연산을 처리할 때 첫번째 bit가 뭔지 알기 위해 31만큼 오른쪽으로 시프트해주었습니다.
2. 첫번째 if, else문
   r포맷과 i와 j포맷으로 나누기 위해 opcode로 구분하였습니다. 왜냐하면 r포맷은 funct로, i포맷과 j포맷은 opcode로 instruction을 알 수 있기 때문입니다. 그래서 if문은 r포맷의 instruction을 위한 것으로, 변수 rd rs rt는 registers의 인덱스로 사용을 했습니다. 각각 연산 중 특히 sra는 첫번째 bit트로 채워야 하는데, 만약 양수라면 sign bit가 0이므로 그저 >>연산만 해주면 되지만, 음수라면 1로 채워야 하므로 INT_MAX를 사용하기위해 limits.h 헤더파일을 추가하였

습니다. 그리고 첫번째 else if는 i포맷과 j포맷인데, 여기서 beq, bne에서 branch address는 pc+offset*4인데, pc주소는 우선 다음으로 넘어간 다음에 offset*4만큼 뛰기 때문에 4를 더해주고 imm에 <<2 연산을 해주었습니다. 그리고 j포맷의 j와 jar에서 jump address는 pc에서 앞의 4비트를 가지고 오기 때문에 0xf0000000와 &연산을 해주었고 ,address 뒤에 00을 붙여야하므로 <<2연산을 하였습니다.

3. 새로 알게 된 점(느낀점)

   이번 pa2는 개인사정으로 시간을 많이 쏟지 못하고 급하게 과제를 수행하게 되었는데, 그만큼 100%로 과제를 수행하지 못한 점이 아쉽습니다. 사실 며칠 밤을 새서 할 수도 있었을텐데,, 반성합니다.