<PA2 REPORT>

1. Process_instruction 함수에서 int instr 를 문자열 bi_instr 에 저장한다. 인수로 받은 instr 가 0xffffffff(halt)면 0 을 반환해야 한다. 아닌 경우 1 을 반환한다. (마지막자리부터 1 과 &해서, '0'에 해당 값만큼 더해 저장한다.) 문자열로 저장된 bi_instr 에서 상위 6 자리는 op 코드로 이 부분을 보고 R,I,J format 을 구분해 처리한다.
    a. R-format : bi_instr 를 끊어가며 순서대로 rs, rt, rd, shamt, funct 라는 int 변수에 저장한다. (strtol 사용) bi_instr 에 문자열을 임시로 각 변수의 문자열에 저장한 후 이를 strtol 로 2 진수로 읽어 int 변수에 저장했다. Add, sub, and, or, nor, sll, srl, jr 은 green sheet 와 동일하게 단순히 연산을 한다. Sra 의 경우 음수의 경우 오른쪽으로 밀고나서 1 로 빈 공간을 채워야 하므로 unsigned int 인 registers 를 signed 로 변환한 후 shift 연산을 수행하고 이를 registers 에 저장한다. Slt 에서는 rs < rt 가 참이면 rd 에 1, 아니면 0 을 저장한다. Registers 는 unsigned 로 선언되어 있어, (signed)로 형변환한 후 비교한다.
    b. I-format : R-format 과 마찬가지로 rs,rt 를 저장하고 나머지 16 자리는 int imm 에 저장한다. Imm 의 최상위비트가 1 이면 ~((1 << 16) – 1)와 or 해서 상위 16 비트를 1 로 만들어준다. Zero extension 도 쓰이기 때문에 32bit 변수 zero_imm 에 imm 을 저장해서 사용했다. Addi 는 green sheet 와 동일하게 작성했다. Andi, ori 에서는 imm 대신 zero_imm 을 사용해 green sheet 대로 작성했다. Lw 는 rs 와 imm 를 더한 unsigned address 변수를 만들고 memory 배열에서 address 에 8bit 씩 읽어와 or 연산으로 합친 값을 rt 에 저장한다. Sw 에서는 반대로 lw 와 동일하게 address 를 선언하고 memory 배열에 address 주소에다 rt 에 8bit 씩 0xff 와 and 한 값을 저장한다. Beq 와 bne 에서는 immediate 값만큼 떨어진 주소로 이동해야 하는데 1word 단위이므로 4 를 곱해준다. (16bit 인 imm 를 32bit 로 확장시키고 << 2 를 해 4 를 곱해준다.) slti 에서도 slt 와 마찬가지로 레지스터 rs 를 signed 로 형변환해주고 비교 후 1 또는 0 을 rt 에 저장한다.
    c. J-format : opcode 이후 26bit 를 strtol 로 2 진수로 읽어 address 변수에 저장한다. 실제 pc 값에 저장해야할 값은 현재 pc 에 상위 4 비트에 26 비트를 sll 2 비트 한 값을 합쳐 저장해야하므로 pc 와 f0000000 를 and 한 값에 address 를 << 2 비트한 값을 or 한 값이다. J 에서는 pc 를 계산한 주소로

업데이트해주고 jal 에서는 현재 pc 값을 레지스터 31(ra)에 저장한 뒤 pc 를 계산한 주로소 업데이트 해준다.

2. Main 함수에서 file 에서 한 줄 씩 처리한 것과 같이 load_program 에서도 file 에서 한 줄 씩 읽어 tokens[0]에 16bit 8 자리가 저장된다. 이 값을 <<를 통해 한 자리씩 읽어 문자열 bi_instr 에 저장한다. 이 크기가 32 인 문자열을 8 개씩 끊어 4 부분으로 저장해 두었다가 memory 에 pc 부터 하나씩 strtol 로 2 진수로 읽어 저장한다. 모두 저장한 뒤 pc 값을 4 증가시켜준다.
3. 실행은 메모리에 8 비트씩 저장된 값을 읽어 와 합친 후 process_instruction 함수에 인수로 전달해주면 된다. 0x00000000 으로 초기화된 word 라는 변수에 memory 에 pc 주소에 있는 값을 자리수에 맞게 쉬프트한 값(24,16,8,0 만큼 <<) word 와 or 연산해 8 비트씩 저장된 값을 32 비트로 합친다. 이후 word 를 process_instruction 함수로 전달해준다. Process_instruction 의 return 이 0 일때까지 반복한다.

Program-hidden

어셈블리를 코드로 변환하면 다음과 같다고 생각한다. 함수 func1, func2 가 있고 처음 func2 가 실행되고 그 안에서 func1 을 호출한다. Func1 을 보면 인수 a0 에는 t1 이 저장되고 a1 은 상위함수에서 3,2,1,0 이 할당된다. Func1 에서는 a1 만큼 a0 를 8 비트씩 오른쪽으로 쉬프트해 0xff 와 and 해 하위 8 비트가 0 인지 확인하고 0 이면 0 을 반환한다.

Func1(ex 0x12345678)	Return 값
A1 = 3	0x12
A1 = 2	0x34
A1 = 1	0x56
A1 = 0	0x78

Func2 에서는 func1 을 a1 을 3,2,1,0 으로 두고 4 번 호출하며 v0 이 0 이면 t0 를 반환한다. 만약 4 번 호출한 후에도 0 이 아니면 t0 를 1 더하고 a0 를 4 더해 다시 처음부터 반복한다. T0 는 func1 이 호출될때마다 증가한다. 호출횟수를 카운트하는 역할을 하는 것 같다. Funct2 는 호출횟수를 카운트하는 t0 를 반복한다. 그리고 코드가 종료된다. 이 program-hidden 은 입력값을 2 진수로 8bit 씩 볼 때 8 비트가 0 인 부분이 있는지 확인하고 없다면 입력값에서 4 를 더해 반복하면서 0 인 부분이 생길때까지 반복횟수를 찾는 프로그램으로 예상된다. Program-hidden 때문에 코드 수정한 부분은 없었고, program- fibonacci 를 해결하니 hidden 도 success 가 나왔다.

```
a0 = s0
func2(a0);
end.

func2(a0){

   t0 = 0;

   while(1){

      a0 save;
      t1 = a0;
      t1 save(t1 = a0);
      a0 = t1;

      for(a1 = 3; a1 >= 0; a1--)
      {
         v0 = func1(a0, a1);
         if(v0 == 0){
            v0 = t0;
            return v0;
         }
         t0++;
         a0 = t1;
      }

      if(v0 != 0){
         t0++;
         a0 load;
         a0 += 4;
      }
   }
}

func1(a0, a1){
   while(a1 != 0){
      a0 = a0 >> 8;
      a1--;
   }
   a0 = a0 & 0xff;
   v0 = a0;
   return v0;
}
```

Figure 1 Program-hidden

새로 알게 된 부분

1. Int16_t 와 int32_t : c 언어에서 int 는 4byte 로 32bit 를 차지하지만 찾아보니 16bit 를 가지는 int 로 명시해 사용할 수 있음을 알게 되었다. Beq,bne 에서 16bits 인 immediate 를 처리할 때 사용했다.
2. Signed int 와 unsigned int : c 언어에서 signed int 와 unsigned int 로 구분해서 선언하고 사용할 경우 unsigned int 에 음수값을 넣으면 매우 큰 양수로 해석되어서 연산이 예상과 달리 이루어진 경우가 있었다. (slt 의 대소비교) unsigned 로 정의된 경우 부호에 주의해야 한다는 것을 알게 됐다.