<PA2 REPORT>

1. Process\_instruction함수에서 int instr를 문자열 bi\_instr에 저장한다. 인수로 받은 instr가 0xffffffff(halt)면 0을 반환해야 한다. 아닌 경우 1을 반환한다. (마지막자리부터 1과 &해서, ‘0’에 해당 값만큼 더해 저장한다.) 문자열로 저장된 bi\_instr에서 상위 6자리는 op코드로 이 부분을 보고 R,I,J format을 구분해 처리한다.
   1. R-format : bi\_instr를 끊어가며 순서대로 rs, rt, rd, shamt, funct라는 int 변수에 저장한다. (strtol사용) bi\_instr에 문자열을 임시로 각 변수의 문자열에 저장한 후 이를 strtol로 2진수로 읽어 int변수에 저장했다. Add, sub, and, or, nor, sll, srl, jr은 green sheet와 동일하게 단순히 연산을 한다. Sra의 경우 음수의 경우 오른쪽으로 밀고나서 1로 빈 공간을 채워야 하므로 unsigned int인 registers를 signed로 변환한 후 shift연산을 수행하고 이를 registers에 저장한다. Slt에서는 rs < rt가 참이면 rd에 1, 아니면 0을 저장한다. Registers는 unsigned로 선언되어 있어, (signed)로 형변환한 후 비교한다.
   2. I-format : R-format과 마찬가지로 rs,rt를 저장하고 나머지 16자리는 int imm에 저장한다. Imm의 최상위비트가 1이면 ~((1 << 16) – 1)와 or해서 상위 16비트를 1로 만들어준다. Zero extension도 쓰이기 때문에 32bit 변수 zero\_imm에 imm을 저장해서 사용했다. Addi는 green sheet와 동일하게 작성했다. Andi, ori에서는 imm대신 zero\_imm을 사용해 green sheet대로 작성했다. Lw는 rs와 imm를 더한 unsigned address변수를 만들고 memory배열에서 address에 8bit씩 읽어와 or연산으로 합친 값을 rt에 저장한다. Sw에서는 반대로 lw와 동일하게 address를 선언하고 memory 배열에address주소에다 rt에 8bit씩 0xff와 and 한 값을 저장한다. Beq와 bne에서는 immediate값만큼 떨어진 주소로 이동해야 하는데 1word단위이므로 4를 곱해준다. (16bit인 imm를 32bit로 확장시키고 << 2를 해 4를 곱해준다.) slti에서도 slt와 마찬가지로 레지스터 rs를 signed로 형변환해주고 비교 후 1 또는 0을 rt에 저장한다.
   3. J-format : opcode이후 26bit를 strtol로 2진수로 읽어 address변수에 저장한다. 실제 pc값에 저장해야할 값은 현재 pc에 상위 4비트에 26비트를 sll 2비트 한 값을 합쳐 저장해야하므로 pc 와 f0000000를 and 한 값에 address를 << 2비트한 값을 or한 값이다. J에서는 pc를 계산한 주소로 업데이트해주고 jal에서는 현재 pc값을 레지스터 31(ra)에 저장한 뒤 pc를 계산한 주로소 업데이트 해준다.
2. Main함수에서 file에서 한 줄 씩 처리한 것과 같이 load\_program에서도 file에서 한 줄 씩 읽어 tokens[0]에 16bit 8자리가 저장된다. 이 값을 <<를 통해 한 자리씩 읽어 문자열 bi\_instr에 저장한다. 이 크기가 32인 문자열을 8개씩 끊어 4부분으로 저장해 두었다가 memory에 pc부터 하나씩 strtol로 2진수로 읽어 저장한다. 모두 저장한 뒤 pc값을 4 증가시켜준다.
3. 실행은 메모리에 8비트씩 저장된 값을 읽어 와 합친 후 process\_instruction함수에 인수로 전달해주면 된다. 0x00000000으로 초기화된 word라는 변수에 memory에 pc주소에 있는 값을 자리수에 맞게 쉬프트한 값(24,16,8,0만큼 <<) word와 or연산해 8비트씩 저장된 값을 32비트로 합친다. 이후 word를 process\_instruction함수로 전달해준다. Process\_instruction의 return이 0일때까지 반복한다.

Program-hidden

어셈블리를 코드로 변환하면 다음과 같다고 생각한다. 함수 func1, func2가 있고 처음 func2가 실행되고 그 안에서 func1을 호출한다. Func1을 보면 인수 a0에는 t1이 저장되고 a1은 상위함수에서 3,2,1,0이 할당된다. Func1에서는 a1만큼 a0를 8비트씩 오른쪽으로 쉬프트해 0xff와 and 해 하위8비트가 0인지 확인하고 0이면 0을 반환한다.

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

Func2에서는 func1을 a1을 3,2,1,0으로 두고 4번 호출하며 v0이 0이면 t0를 반환한다. 만약 4번 호출한 후에도 0이 아니면 t0를 1더하고 a0를 4더해 다시 처음부터 반복한다. T0는 func1이 호출될때마다 증가한다. 호출횟수를 카운트하는 역할을 하는 것 같다. Funct2는 호출횟수를 카운트하는 t0를 반복한다. 그리고 코드가 종료된다. 이 program-hidden은 입력값을 2진수로 8bit씩 볼 때 8비트가 0인 부분이 있는지 확인하고 없다면 입력값에서 4를 더해 반복하면서 0인 부분이 생길때까지 반복횟수를 찾는 프로그램으로 예상된다. Program-hidden때문에 코드 수정한 부분은 없었고, program- fibonacci를 해결하니 hidden도 success가 나왔다.

새로 알게 된 부분

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;

}

1. Int16\_t와 int32\_t : c언어에서 int는 4byte로 32bit를 차지하지만 찾아보니 16bit를 가지는 int로 명시해 사용할 수 있음을 알게 되었다. Beq,bne에서 16bits인 immediate를 처리할 때 사용했다.
2. Signed int 와 unsigned int : c 언어에서 signed int와 unsigned int로 구분해서 선언하고 사용할 경우 unsigned int에 음수값을 넣으면 매우 큰 양수로 해석되어서 연산이 예상과 달리 이루어진 경우가 있었다. (slt의 대소비교) unsigned로 정의된 경우 부호에 주의해야 한다는 것을 알게 됬다.

Figure Program-hidden