Lab report #2

CSED 20170302 Kim Dae Hui

What did you do last week?

지난 주 금요일 토요일날 포카전이 있었는데, 이번 포카전에 축구선수단으로 출전하게 되어서 8월 1일부터 지난 주 수요일까지 쉼없는 훈련이 계속되었다. 정말 힘들었다. 매일밤 6시부터 10시까지 월,수,금 아침6시에 일어나서 아침훈련도 계속해서 사실 공부할 틈이 없었다. 그래서 lab퀴즈도 정말 못본 것 같다… 공부를 안해서 뭐 알 수 있는것도 없었고 사실 수업때도 하필 월요일 수요일 이 과목이 있어서 많이 졸았다… 그렇게 포카전이 금요일날 끝나고 열심히 했지만 축구는 연장전 후반 추가시간에 역전골을 먹으면서 패배했다.. 정말 운이 없던 주간인것 같다. 아직도 골 들어가는게 눈에 훤하게 그려진다. 그 후 축구선수단 뒷풀이를 갖고 집에 내려가서 쉬다가 와서 월요일부터 부랴부랴 공부해서 겨우겨우 마무리 랩을 어찌저찌 마무리 짓게 되었다. 코딩을 하면서 너무 직관적으로 한게 아닌가 정말 내가 생각해도 무식하게 짯다 싶은것들도 많고 시간이 더 많았으면 하는 아쉬움이 크다. 앞으로 창훈이형한테 SMP배우면서 공부열심히해서 앞에것도 따라잡고 뒤에 lab시간도 다 잘할 수 있기를 빈다. 대부분 디지털 시스템 설계나 수학과목에서 배운 내용들을 사용해서 카노맵이나 퀸맥등 방법으로 직관적으로 해결한 것들이다…

1.

/\*

\* bitOr - x|y using only ~ and &

\* Example: bitOr(6, 5) = 7

\* Legal ops: ~ &

\* Max ops: 8

\* Rating: 1

\*/

int bitOr(int x, int y) {

return ~ (~x&~y);

}

디지털 시스템설계 수업에서 배운 논리연산과 드모르간의 법칙을 생각하여 A합B를 (A교집합B)’로 표현한 것을 코드로 표현하였다.

2.

/\*

\* logicalShift - shift x to the right by n, using a logical shift

\* Can assume that 0 <= n <= 31

\* Examples: logicalShift(0x87654321,4) = 0x08765432

\* Legal ops: ! ~ & ^ | + << >>

\* Max ops: 20

\* Rating: 3

\*/

int logicalShift(int x, int n) {

return (x>>n) & ~(((1<<31)>>n)<<1);

} 우선 기본적으로 arithmetic shift가 발생하는 점을 이용하여 0x01인 1을 생성한 후 left shift를 이용하여 맨 첫자리 sign자리에 넣어준다. 그 후 이를 n만큼 shift해주고 1이 원래 차지하고 있던 한자리 만큼을 다시 left shift시키면 n칸만큼 (1<<31)이 right shift한 상태가 된다. 이를 negation하면 그 상태의 MSB부터 n칸만큼의 bit가 0값이 오고 그와 원래대로 shift한 수를&연산해주면 logical shift로 된다.

3.

/\*

\* bitCount - returns count of number of 1's in word

\* Examples: bitCount(5) = 2, bitCount(7) = 3

\* Legal ops: ! ~ & ^ | + << >>

\* Max ops: 40

\* Rating: 4

\*/

int bitCount(int x) {

int mask=0x11;

int result,checker;

mask = mask|(mask<<8);

mask = mask|(mask<<16);

result = x&mask;

result = result + (mask&(x>>1));

result = result + (mask&(x>>2));

result = result + (mask&(x>>3));

result = result + (result>>16);

checker=0xf|(0xF<<8);

result = (result& checker) + ((result>>4) & checker);

return ((result + (result >> 8)) & 0x3f);

}먼저 다른 친구들이 많이들 mask나 sign이라는 표현을 쓰는데 좋은 것 같아서 원래 변수이름은 isone이였지만 mask로 바꿔보았다. 0x11으로 0000 0000 0000 0000 0000 0000 0001 0001 인 mask를 생성하고 이 후 이것들을 다시 더하고 재조합해서 0001 0001 0001 0001 0001 0001 0001 0001인 상태로 만든다. 그 후 bitCount받길 원하는 변수인 x를 &연산자로 검사해서 지금 1이있는 자리의 1의 갯수를 센다. 그리고 1자리씩 옮겨가며 000부분에 있던 1들도 세어준다. 마지막에 0xf수를 생성하여 mask와 마찬가지로 재조합하여 1의 갯수를 셀 수 있도록 checker를 생성한 후 checker를 4bit씩 right shift시키면서 result에 1의 갯수를 count한다. 0x3f까지 &로 검사해서 마지막 MSB자리가 shift될때까지 검사해준다.

4.

/\*

\* negate - return -x

\* Example: negate(1) = -1.

\* Legal ops: ! ~ & ^ | + << >>

\* Max ops: 5

\* Rating: 2

\*/

int negate(int x) {

return ~x +1;

} negation은 2’s complement를 만드는 과정과 똑같이 생각하여 1과 0을 반전시켜준 후에 0x01인 1을 더해주면 된다.

5.

/\*

\* addOK - Determine if can compute x+y without overflow

\* Example: addOK(0x80000000,0x80000000) = 0,

\* addOK(0x80000000,0x70000000) = 1,

\* Legal ops: ! ~ & ^ | + << >>

\* Max ops: 20

\* Rating: 3

\*/

int addOK(int x, int y)

{

int result;

result = (x^y)|(~((x+y)^x));

result = result>>31;

result = result&1;

return result;

}add OK는 MSB자리의 carry의 값을 계산해줘서 overflow를 감지해주면 된다. Carry가 1이면은 0을, 아니면1을 return해주면 된다. 이를 카노맵으로 나타낸 후 퀸-맥을 이용해서 식을 간소화시키면 result = (x^y)|(~((x+y)^x))이 결과가 나온다. 그 후 MSB를 >>31한 후 1과 &연산을통해 carry가 있는지 없는지알아낼 수 있다.

6.

/\*

\* isLessOrEqual - if x <= y then return 1, else return 0

\* Example: isLessOrEqual(4,5) = 1.

\* Legal ops: ! ~ & ^ | + << >>

\* Max ops: 24

\* Rating: 3

\*/

int isLessOrEqual(int x, int y) {

int nx =~x+1;

int yadd =nx + y;

int checker =yadd>>31 &1;

int leftside=1<<31;

int xl = leftside&x;

int yl = leftside&y;

int bomb = (xl^yl);

bomb =(bomb>>31)&1;

return (bomb & (xl>>31)) | (!checker & !bomb);

}x가 항상 y보다 작거나 같아야하기 때문에 무조건 x-y는 음수가 나와야 한다. 그 원리로 항상 y-x는 양수이기에 MSB자리는 0으로 채워져 있어야 한다. 그래서 checker를 이용해서 x가 y보다 작은지 알아내고 bomb와 xㅣ 과 yㅣ을 이용해서 MSB를 검사하고 같은수를 판별하고 작음을 확인함.

7.

/\*

\* float\_neg - Return bit-level equivalent of expression -f for

\* floating point argument f.

\* Both the argument and result are passed as unsigned int's, but

\* they are to be interpreted as the bit-level representations of

\* single-precision floating point values.

\* When argument is NaN, return argument.

\* Legal ops: Any integer/unsigned operations incl. ||, &&. also if, while

\* Max ops: 10

\* Rating: 2

\*/

unsigned float\_neg(unsigned uf) {

int Nanchecker = 0x000000FF<<23;

int frac = 0x7FFFFF & uf;

if((Nanchecker&uf)==Nanchecker&&frac)

return uf;

return uf^(1<<31);

} not a number인 경우인 8자리가 1로 채워진 경우 nanchecker로 바로 return해주고 frac부분을 계산하여 return할때 MSB자리를 exclusive or 연산으로 처리한다. 그래서 부호를 바꿔준다.

8.

/\*

\* float\_i2f - Return bit-level equivalent of expression (float) x

\* Result is returned as unsigned int, but

\* it is to be interpreted as the bit-level representation of a

\* single-precision floating point values.

\* Legal ops: Any integer/unsigned operations incl. ||, &&. also if, while

\* Max ops: 30

\* Rating: 4

\*/

unsigned float\_i2f(int x) {

int e=158;

int sign = 1<<31;

int xsign = x&sign;

int frac;

if(x==sign) return sign|(158<<23);

if (!x) return 0;

if(xsign)

x = ~x+1;

while(!(x&sign))

{

x=x<<1;

e=e-1;

}

frac = (x&(~sign)) >> 8;

if (x&0x80 && ((x&0x7F) > 0 || frac&1)) frac=frac+1;

return xsign + (e<<23) + frac;

}integer를 비트연산으로 float으로 변환시키는 규칙을 따른다. X의 MSB를 xsign으로 관리하여 float부분으로 관리하고 남은 부분을 frac으로 변환하여 float화 시키는 규칙을 따른다.

9.

/\*

\* float\_twice - Return bit-level equivalent of expression 2\*f for

\* floating point argument f.

\* Both the argument and result are passed as unsigned int's, but

\* they are to be interpreted as the bit-level representation of

\* single-precision floating point values.

\* When argument is NaN, return argument

\* Legal ops: Any integer/unsigned operations incl. ||, &&. also if, while

\* Max ops: 30

\* Rating: 4

\*/

unsigned float\_twice(unsigned uf) {

if(uf==0)return uf;

if(((uf>>23)&0xff)==0xff) return uf;

if(((uf>>23)&0xff)==0x00) return (uf&(1<<31))|(uf<<1);

return uf + (1<<23);

} 0의 값이 들어오면은 그리고 not a number일경우에도 그냥 그대로 리턴해주고 만약에 그게 아닌 경우에는 (uf&(1<<31))|(uf<<1)로 2배해준다. 그 외의 경우는 그냥 uf + (1<<23);을 통해 2배해준다.