

Datalab

비트연산 & Datalab

2016.09.27

황슬아

seula.hwang@cnu.ac.kr

목차

1. 개요
2. Boolean Algebra
3. 비트 논리연산자
4. 비트 연산자
5. Datalab

개요

1. 실습명

- ✓ 비트 연산 & Datalab

2. 목표

- ✓ 비트 연산의 이해와 사용

3. 주제

- ✓ 비트 연산
 - Boolean Algebra
 - 비트 논리 연산자
 - 비트 연산자
- ✓ Datalab

Boolean Algebra

1. 불 대수(Boolean Algebra)

- ✓ 논리를 표현하기 위한 Algebra(대수학)이며, 컴퓨터 내부의 비트 정보 표시에 사용하는 변수들은 0 또는 1을 가지며, 2진수의 표시 및 연산에 매우 유용하다.

❖ AND (&)

❖ $A \& B = 1$ when both $A = 1$ and $B = 1$

&	0	1
0	0	0
1	0	1

❖ OR (|)

❖ $A | B = 1$ when either $A = 1$ and $B = 1$

	0	1
0	0	1
1	1	1

❖ NOT (~)

❖ $\sim A = 1$ when $A = 0$

~	0
0	1
1	0

❖ Exclusive OR (XOR, ^)

❖ $A \wedge B = 1$ when either $A = 1$ or $B = 1$, but not both

^	0	1
0	0	1
1	1	0

비트 논리 연산자

1. Bit 단위 논리 연산에 사용하는 연산자

- 1) $\&$, $|$, \sim , \wedge
- 2) 일반 논리 연산자($\&\&$, $||$, $!$) 보다 우선순위가 높음
- 3) 아래의 정수형 변수에 사용 가능
 - int, short, long, char, unsigned

2. 예제 (8-bit)

- 1) $0x69 \& 0x55 = 0x41$
 - $0110\ 1001_{(2)} \& 0101\ 0101_{(2)} = 0100\ 0001_{(2)}$
- 2) $0x69 | 0x55 = 0x7D$
 - $0110\ 1001_{(2)} | 0101\ 0101_{(2)} = 0111\ 1101_{(2)}$
- 3) $\sim 0x41 = 0xBE$
 - $\sim 0100\ 0001_{(2)} = 1011\ 1110_{(2)}$
- 4) $0x41 \wedge 0x69 = 0x28$
 - $0100\ 0001_{(2)} \wedge 0110\ 1001_{(2)} = 0010\ 1000_{(2)}$

비트 연산자(1/2)

1. Shift 연산자 (<<, >>)

- ✓ 어셈블리 언어나 기계어의 프로그램 작성에서 레지스터 또는 기억 장소 내에 비트 값들을 왼쪽이나 오른쪽으로 이동시키는 것.

2. Left shift ($x \ll y$)

- ✓ Shift bit-vector x left y position
 - 좌측 끝의 MSB를 날려 버림
 - 우측 끝의 LSB에 0을 채워줌
- ✓ Logical shift와 Arithmetic shift의 동작이 동일함

비트 연산자(2/2)

3. Right shift ($x \gg y$)

- 1) Shift bit-vector x right y position
 - ✓ 우측 끝의 LSB를 날려버림
- 2) Logical shift (논리적 자리 이동)
 - ✓ 좌측 끝의 MSB에 0을 채워줌
- 3) Arithmetic shift (산술 자리 이동)
 - ✓ 좌측 끝의 MSB에 현재 부호를 유지함
 - ✓ Two's complement 정수 표현 시에 필요

Argument x	1010 0010
$x \ll 3$	0001 0000
Logical $x \gg 2$	0010 1000
Arithmetic $x \gg 2$	1110 1000

기본 비트 연산 – 따라 하기

1. /home/ubuntu/lab03/operation_test.tar.gz를 자신의 홈(~)으로 복사
2. 압축 해제 후 디렉터리 내 파일 확인

 sys00@localhost: ~

```
sys00@localhost:~$ cp /home/ubuntu/lab03/operation_test.tar.gz ~
sys00@localhost:~$ ls
operation_test.tar.gz
sys00@localhost:~$ tar xvfz operation_test.tar.gz
operation_test/
operation_test/RShiftFunc.c
operation_test/Makefile
operation_test/AndFunc.c
operation_test/XorFunc.c
operation_test/InputAndPrint.c
operation_test/OrFunc.c
operation_test/LShiftFunc.c
operation_test/NotFunc.c
operation_test/header.h
sys00@localhost:~$
```


기본 비트 연산 – 따라 하기

3. 각 소스 내부의 함수를 작성

AndFunc.c (~operation_test) - VIM

```
1 int AndFunc(int nA, int nB){
2
3     int result = 0;
4     //nA와 nB를 bit단위로 AND 연산하여 반환하시오.
5     result = ;
6     return result;
7 }
8
```

4. make를 통해 컴파일 후 결과를 확인

```
sys00@localhost:~/operation_test$ make
gcc -o test.out -g -O2 AndFunc.c LShiftFunc.c NotFunc.c RShiftFunc.c InputAndPrint.c OrFunc.c XorFunc.c
InputAndPrint.c: In function 'main':
InputAndPrint.c:8:7: warning: ignoring return value of 'scanf', declared with attribute warn_unused_result [-Wunused-result]
    scanf("%x%x", &nA, &nB);
    ^
sys00@localhost:~/operation_test$ ls
AndFunc.c  InputAndPrint.c  Makefile  OrFunc.c  test.out
header.h   LShiftFunc.c     NotFunc.c RShiftFunc.c XorFunc.c
sys00@localhost:~/operation_test$
```

소스 코드	코드
AndFunc.c	nA & nB
NotFunc.c	~nA
OrFunc.c	nA nB
XorFunc.c	nA ^ nB
RShiftFunc.c	nA >> nB
LShiftFunc.c	nA << nB

Datalab

- 정수와 실수의 bit-level 표현에 대해 좀 더 친숙해 질 수 있도록 만들어진 랩
 - 1) 현 실습에서는 정수의 bit-level에 대한 표현만 진행
 - 2) 각 문제마다 사용 가능한 연산자들을 가지고 코드를 작성
 - 해당 연산자는 다음 슬라이드에서 확인


Datalab – 함수 설명

- Datalab은 아래와 같이 8개의 함수로 이루어져있다.

함수	기능	사용 가능한 연산자
bitAnd(int x, int y)	~와 을 사용해서 x & y 연산	~
getByte(int x, int n)	x의 n번째 (1 byte)를 추출	! ~ & ^ + << >>
logicalShift(int x, int n)	logicalShift를 이용해서 오른쪽으로 shift	! ~ & ^ + << >>
bitCount(int x)	x에서 1의 개수를 계산	! ~ & ^ + << >>
isZero(int x)	x == 0 이면 1 아니면 0	! ~ & ^ + << >>
isEqual(int x, int y)	x == y 이면 1 아니면 0	! ~ & ^ + << >>
fitsBits(int x, int n)	n-bit로 x의 2의 보수를 표현할 수 있으면 1을 반환	! ~ & ^ + << >>
isLessOrEqual(int x, int y)	x ≤ y 이면 1 아니면 0	! ~ & ^ + << >>
rotateLeft(int x, int n)	x를 n만큼 오른쪽으로 회전	~ & ^ + << >> !

Datalab – 구조

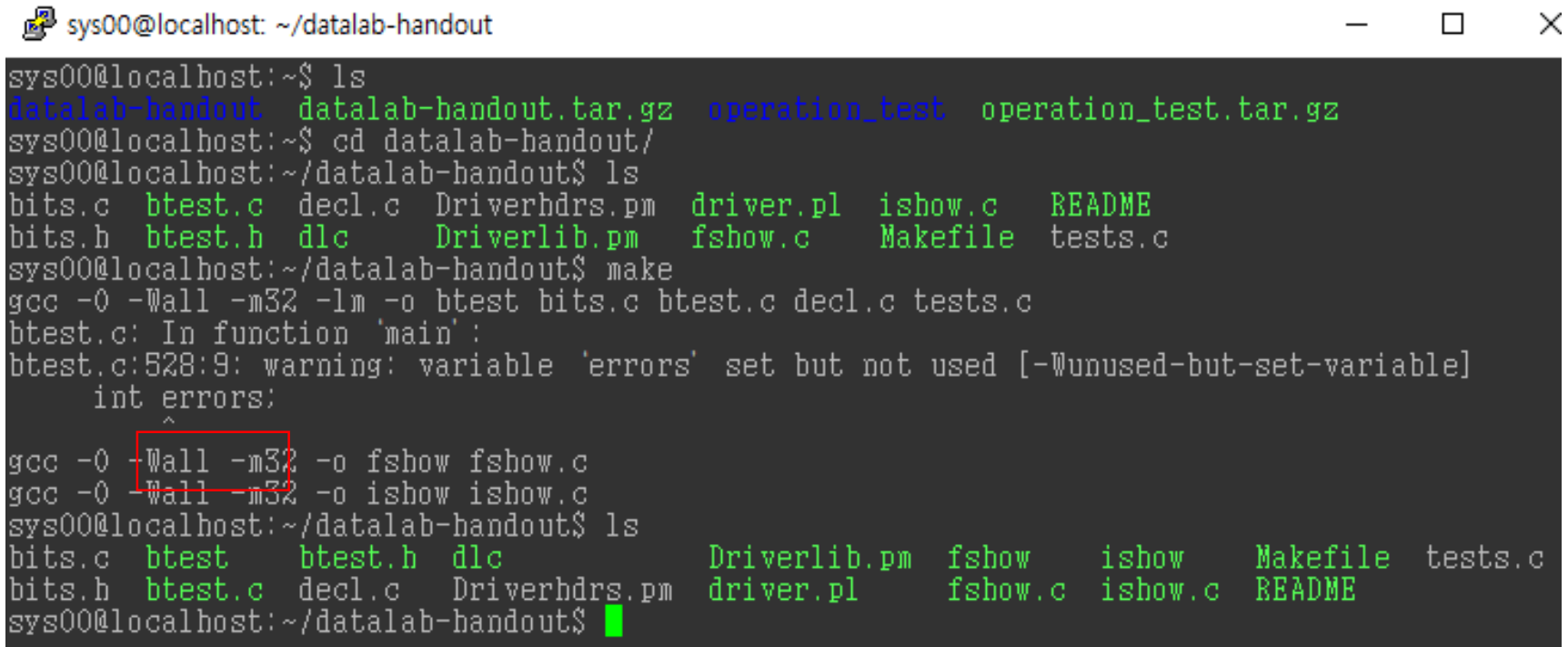
1. /home/ubuntu/Datalab/datalab-handout.tar.gz 파일을 자신의 계정에 복사 후 압축 해제
2. 실습 소스코드 작성은 **bits.c** 파일에 한다.

 sys00@localhost: ~

```
sys00@localhost:~$ cp /home/ubuntu/Datalab/datalab-handout.tar.gz ~
sys00@localhost:~$ ls
datalab-handout.tar.gz  operation_test  operation_test.tar.gz
sys00@localhost:~$ tar xvfz datalab-handout.tar.gz
datalab-handout/
datalab-handout/Makefile
datalab-handout/driver.pl
datalab-handout/README
datalab-handout/bits.h
datalab-handout/Driverhdrs.pm
datalab-handout/ishow.c
datalab-handout/fshow.c
datalab-handout/btest.c
datalab-handout/decl.c
datalab-handout/bits.c
datalab-handout/tests.c
datalab-handout/dlc
datalab-handout/Driverlib.pm
datalab-handout/btest.h
sys00@localhost:~$ █
```

Datalab – 컴파일

1. datalab-handout 디렉터리에서 make 명령을 통해 컴파일 하면 실행파일 **btest**가 생성됨



```
sys00@localhost: ~/datalab-handout
sys00@localhost:~$ ls
datalab-handout  datalab-handout.tar.gz  operation_test  operation_test.tar.gz
sys00@localhost:~$ cd datalab-handout/
sys00@localhost:~/datalab-handout$ ls
bits.c  btest.c  decl.c  Driverhdrs.pm  driver.pl  ishow.c  README
bits.h  btest.h  dlc      Driverlib.pm   fshow.c   Makefile  tests.c
sys00@localhost:~/datalab-handout$ make
gcc -O -Wall -m32 -lm -o btest bits.c btest.c decl.c tests.c
btest.c: In function 'main':
btest.c:528:9: warning: variable 'errors' set but not used [-Wunused-but-set-variable]
    int errors;
        ^
gcc -O -Wall -m32 -o fshow fshow.c
gcc -O -Wall -m32 -o ishow ishow.c
sys00@localhost:~/datalab-handout$ ls
bits.c  btest  btest.h  dlc      Driverlib.pm  fshow  ishow  Makefile  tests.c
bits.h  btest.c  decl.c   Driverhdrs.pm driver.pl     fshow.c ishow.c README
sys00@localhost:~/datalab-handout$
```

Datalab – 진행방법

1. vi 편집기를 이용하여 bits.c 파일을 열어 맨 위에 자신의 이름과 학번을 기입한다.

bits.c + (~/datalab-handout) - VIM

```
1  /*
2  * NAME: Hello world
3  * NUMBER: 201600000
4  *
5  */
6
7  /*
8  * CS:APP Data Lab
9  *
10 * <Please put your name and userid here>
11 *
12 * bits.c - Source file with your solutions to the Lab.
13 *          This is the file you will hand in to your instructor.
```

2. 함수를 문제에 알맞게 작성 후 저장한다. make를 사용해 다시 컴파일 해서 실행파일 생성

Datalab – 주의

1. 컴파일 후, 반드시 자신이 허용된 연산자를 사용해 코드를 작성하였는지 확인
2. ./dlc bits.c
 - 1) 해당 명령어를 사용하면 자신의 코드에서 허용되지 않은 연산자가 사용되었는지 확인할 수 있음
 - 2) 허용되지 않은 연산자를 사용하면, 해당 문제 점수 감점
 - Ex) for, while 과 같은 구문 등
3. bits.c 외 다른 파일 수정 금지

Datalab – 결과확인

1. 함수를 작성한 후 ./btest를 통해 점수를 확인

✓ bits.c를 수정하였을 경우 make를 통해 다시 컴파일 해주어야만 수정된 결과가 반영된다.

```
sys02@localhost:~/datalab-handout$ ./btest
Score  Rating  Errors  Function
1      1       0     bitAnd
2      2       0     getByte
3      3       0     logicalShift
4      4       0     bitCount
1      1       0     isZero
2      2       0     isEqual
2      2       0     fitsBits
3      3       0     isLessOrEqual
3      3       0     rotateLeft
Total points: 21/21
```

2. 특정 함수의 점수만 확인 하는 방법

- 1) -f 옵션을 사용하여 함수 별로 결과를 확인할 수 있음
- 2) 사용법: ./btest -f [함수명]

```
sys00@localhost:~/datalab-handout$ ./btest -f bitAnd
Score  Rating  Errors  Function
1      1       0     bitAnd
Total points: 1/1
```


Datalab – 결과확인

3. 정답이 아닌 경우 다음과 같은 에러 발생

```
sys00@localhost:~/datalab-handout$ ./btest
Score  Rating  Errors  Function
ERROR: Test bitAnd(-2147483648[0x80000000],-2147483648[0x80000000]) failed...
...Gives 2[0x2]. Should be -2147483648[0x80000000]
ERROR: Test getByte(-2147483648[0x80000000],0[0x0]) failed...
...Gives 2[0x2]. Should be 0[0x0]
ERROR: Test logicalShift(-2147483648[0x80000000],0[0x0]) failed...
...Gives 2[0x2]. Should be -2147483648[0x80000000]
ERROR: Test bitCount(-2147483648[0x80000000]) failed...
...Gives 2[0x2]. Should be 1[0x1]
ERROR: Test isZero(-2147483648[0x80000000]) failed...
...Gives 2[0x2]. Should be 0[0x0]
ERROR: Test isEqual(-2147483648[0x80000000],-2147483648[0x80000000]) failed...
...Gives 2[0x2]. Should be 1[0x1]
ERROR: Test fitsBits(-2147483648[0x80000000],1[0x1]) failed...
...Gives 2[0x2]. Should be 0[0x0]
ERROR: Test isLessOrEqual(-2147483648[0x80000000],-2147483648[0x80000000]) failed...
...Gives 2[0x2]. Should be 1[0x1]
ERROR: Test rotateLeft(-2147483648[0x80000000],0[0x0]) failed...
...Gives 2[0x2]. Should be -2147483648[0x80000000]
Total points: 0/21
sys00@localhost:~/datalab-handout$ vi bit.c
sys00@localhost:~/datalab-handout$ vi bits.c
sys00@localhost:~/datalab-handout$ vi bits.c
sys00@localhost:~/datalab-handout$ make
gcc -O -Wall -m32 -lm -o btest bits.c btest.c decl.c tests.c
btest.c: In function 'main':
btest.c:528:9: warning: variable 'errors' set but not used [-Wunused-but-set-variable]
    int errors;
        ^
sys00@localhost:~/datalab-handout$ ./btest
```

Datalab – 종합 결과 확인

- 종합적인 점수를 확인하기 위해서는 ./driver.pl 파일 실행
 - ✓ 실행 전 chmod 700 driver.pl을 해서 사용자에게 해당 파일의 모든 권한을 준다.

```
5. Running './dlc -e' to get operator count of each function.
```

Correctness Results			Perf Results		
Points	Rating	Errors	Points	Ops	Puzzle
0	1	1	0	0	bitAnd
0	2	1	0	0	getByte
0	3	1	0	0	logicalShift
0	4	1	0	0	bitCount
0	1	1	0	0	isZero
0	2	1	0	0	isEqual
0	2	1	0	0	fitsBits
0	3	1	0	0	isLessOrEqual
0	3	1	0	0	rotateLeft

```
Score = 0/39 [0/21 Corr + 0/18 Perf] (0 total operators)  
sys02@localhost:~/datalab-handout$
```

Datalab – 점수 측정 기준

- 점수 평가
 - ✓ 총 39점 만점
 - 정확성 (Correctness): 21점
 - 성능 (Performance): 18점
 - ✓ 정확성 평가표 (난이도에 따라 점수가 다름)

함수	점수
bitAnd(int x, int y)	1
getByte(int x, int n)	2
logicalShift(int x, int n)	3
bitCount(int x)	4
isZero(int x)	1
isEqual(int x, int y)	2
fitsBits(int x, int n)	2
isLessOrEqual(int x, int y)	3
rotateLeft(int x, int n)	3

과제

1. Datalab의 bits.c와 보고서를 하나의 파일로 압축하여 사이버캠퍼스에 제출(추가적으로 보고서를 출력하여 서면 제출)

1) PDF 파일로 제출

- 한글과 MS word의 다른 이름으로 저장 기능 활용

2) 파일 제목: [sys00]datalab_학번_이름

3) 반드시 위의 양식을 지켜야 함. (위반 시 감점)

4) 보고서는 제공된 양식 사용

2. COPY 시 0점 처리

3. 제출일자

1) 사이버 캠퍼스: 2016년 10월 04일 화요일 8시 59분 59초까지

2) 서면 제출 : 2016년 10월 04일 실습시간까지