Marcin Stępniak

Architektura systemów komputerowych
Laboratorium 13
Symulator SMS32
Operacje na bitach

1. Informacje

Matematyk o nazwisku Bool wymyślił gałąź matematyki do przetwarzania wartości „prawda" i „fałsz" zamiast liczb. Gałąź ta została nazwana algebrą Boole'a. Prosta algebra Boole'a jest zgodna z intuicją, ale jeśli trzeba przetwarzać decyzje wykorzystujące wiele wartości, które mogą być prawdziwe lub fałszywe, według skomplikowanych zasad, potrzebna jest ta gałąź matematyki.

1.1. Operacja AND

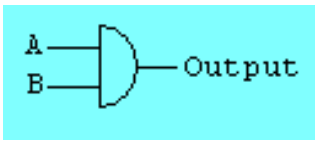
Na obu wejściach musi być prawda, aby na wyjściu była prawda. Operacja AND służy do dodawania i podejmowania decyzji.

Tabela 1. Tablica prawdy dla operacji AND

A	B	wyjście
0	0	0
0	1	0
1	0	0
1	1	1

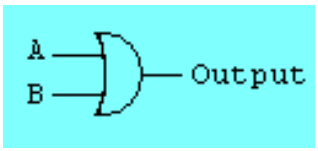
1.2. Operacja OR

Oba wejścia muszą być fałszywe aby na wyjściu pojawił się fałsz. Operator OR jest stosowany w procesie podejmowania decyzji. Zarówno AND



Rysunek 1. Bramka AND

jak i OR służą do maskowania bitów. Maskowanie bitów służy do wybierania poszczególnych bitów w bajcie lub do ich ustawiania. OR wykorzystuje się do ustawiania odpowiednich bitów na wartość „1”. AND jest używany do ustawiania bitów na „0”. AND jest wykorzystywany także do sprawdzenia czy dany bit jest jedynką. OR służy do sprawdzania, czy bit jest zerem.



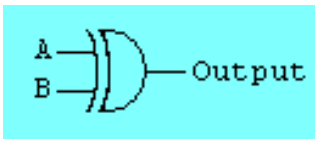
Rysunek 2. Bramka OR

Tabela 2. Tablica prawdy dla operacji OR

A	B	wyjście
0	0	0
0	1	1
1	0	1
1	1	1

1.3. Operacja XOR

Jeżeli na bitach obrazu graficznego wykona się operację XOR z innymi bitami, to pojawia się nowy obraz. Jeśli powtórzy się operację XOR, to ten obraz znika. Tak dzieje się w przypadku kursora myszy przesuwanego po ekranie. XOR w połączeniu z AND stosuje się przy operacji dodawania. XOR wykrywa, czy wejścia są równe, czy nie.



Rysunek 3. Bramka XOR

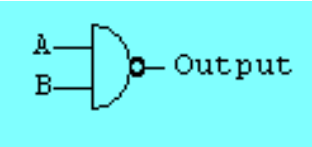
Tabela 3. Tablica prawdy dla operacji XOR

A	B	wyjście
0	0	0
0	1	1
1	0	1
1	1	0

1.4. Operacja NAND

NAND jest w rzeczywistości zanegowanym wyjściem operacji AND. Układy elektroniczne często używają bramek NAND, ale języki programowania (wliczając symulator SMS32) nie udostępniają operacji NAND. Zamiast niej należy używać NOT AND. W symulatorze może to wyglądać np. tak:

```
AND AL, BL
NOT AL
```



Rysunek 4. Bramka NAND

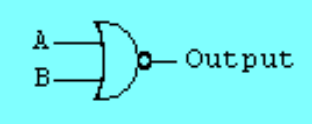
Tabela 4. Tablica prawdy dla operacji NAND

A	B	wyjście
0	0	1
0	1	1
1	0	1
1	1	0

1.5. Operacja NOR

NOR jest zanegowanym wyjściem operacji OR. Podobnie jak w przypadku NAND, układy elektroniczne często wykorzystują bramkę NOR, ale operacja NOR nie jest dostępna w językach programowania. Zamiast niej należy używać NOT OR. W symulatorze może to wyglądać np. tak:

```
OR AL, BL
NOT AL
```



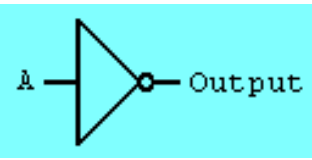
Rysunek 5. Bramka NOR

Tabela 5. Tablica prawdy dla operacji NOR

A	B	wyjście
0	0	1
0	1	0
1	0	0
1	1	0

1.6. Operacja NOT

Operacja NOT neguje wszystkie bity w bajcie. Każda „1” zostaje zamieniona na „0”, a każde „0” na „1”.



Rysunek 6. Bramka NOT

Tabela 6. Tablica prawdy dla operacji NOT

A	wyjście
0	1
1	0

1.7. Operacja SHL

Operacja SHL przesuwa bity w lewo. Najbardziej znaczący bit zostaje odrzucony, pozostałe bity zostają przesunięte o jedną pozycję w lewo, a najmniej znaczący bit przyjmuje wartość „0”.

1.8. Operacja SHR

Przesuwa bity w prawo. Najmniej znaczący bit zostaje odrzucony, pozostałe bity zostają przesunięte o jedną pozycję w prawo, a najbardziej znaczący bit przyjmuje wartość „0”.

1.9. Operacja ROL

Działa podobnie do SHL. Przesuwa bity w lewo, ale najbardziej znaczący bit nie jest odrzucany. Trafia on w miejsce najmniej znaczącego bitu.

1.10. Operacja ROR

Działa podobnie do SHR. Przesuwa bity w prawo, ale najmniej znaczący bit nie jest odrzucany. Trafia on w miejsce najbardziej znaczącego bitu.

Tabela 7. Instrukcje dla operacji bitowych

Instr.	Operand A	Operand B	Działanie
AND	rejestr ogólnego przeznaczenia	konkretna wartość lub rejestr	Wykonuje operację A AND B. Wynik zapisuje w A.
OR	rejestr ogólnego przeznaczenia	konkretna wartość lub rejestr	Wykonuje operację A OR B. Wynik zapisuje w A.
XOR	rejestr ogólnego przeznaczenia	konkretna wartość lub rejestr	Wykonuje operację A XOR B. Wynik zapisuje w A.
NOT	rejestr ogólnego przeznaczenia		Wykonuje negację (odwraca wszystkie bity) na rejestrze.
ROL	rejestr ogólnego przeznaczenia		Obraca bity w rejestrze w lewo.
ROR	rejestr ogólnego przeznaczenia		Obraca bity w rejestrze w prawo.
SHL	rejestr ogólnego przeznaczenia		Przesuwa bity w rejestrze w lewo.
SHR	rejestr ogólnego przeznaczenia		Przesuwa bity w rejestrze w prawo.

Listing 1. Operacje na bitach

```
   mov al,80
loop:
   shr al
   push al
   pop bl
   and bl, 8; sprawdzenie jedynki na czwartej pozycji od prawej
   jz loop
   or al, 2 ; ustawienie bity na drugiej pozycji od prawej
loop2:
   rol al ; obrot bitow w lewo
   push al
   pop bl
   and bl, 1; sprawdzenie jedynki na pierwszej pozycji
   jz loop2
end
```

1.11. Dodatkowe informacje

1. http://www.softwareforeducation.com/sms32v50/sms32v50_manual/380-TruthTables.htm
2. https://pl.wikipedia.org/wiki/Algebra_Boole'a
3. https://pl.wikipedia.org/wiki/Kod_Graya
4. http://www.linux.net.pl/~wkotwica/megipt3.html

2. Zadania

2.1. Zadanie 1

Napisać program ustawiający flagę S (znaku). Program może wykorzystywać tylko instrukcje PUSHF, POPF, PUSH, POP, OR I END.

2.2. Zadanie 2

Napisać program płynnie sterujący silnikiem krokowym (bez zmian w szybkości). Program powinien działać w pętli nieskończonej. Poza numerem portu, kod programu musi zawierać tylko jeden bajt danych.

2.3. Zadanie 3

Napisać program, który zamienia liczbę w systemie U2 na liczbę przeciwną, np. 23 na -23 i odwrotnie. Należy wykorzystać operacje bitowe.

2.4. Zadanie 4

Napisać program zamieniający liczbę w naturalnym kodzie binarnym na kod Graya i odwrotnie. Do konwersji w każdą stronę należy zastosować oddzielną procedurę.

Listing 2. Algorytm konwersji na/z kody Graya

```
/*
 * Funkcja konwertuje bajt liczby w NKB
 * na kod Graya.
 * Operator >> jest operatorem przesuniecia
 * w prawo o wskazana liczbe bitow.
 * Operator ^ to XOR
 */
byte binaryToGray(byte num)
{
    return num ^ (num >> 1);
}
```

```
/*
 * Zamiana z kodu Graya na liczbe NKB.
 */
byte grayToBinary(byte num)
{
    num = num ^ (num >> 4);
    num = num ^ (num >> 2);
    num = num ^ (num >> 1);
    return num;
}
```

2.5. Zadanie 5

Napisać program wykonujący mnożenie według algorytmu starożytnych Egipcjan (zob. listing 3)

Listing 3. Algorytm mnożenia starożytnych Egipcjan

```
c = 0
while b != 0
    if (b and 1) != 0
        c = c + a
    left shift a by 1
    right shift b by 1
return c
```