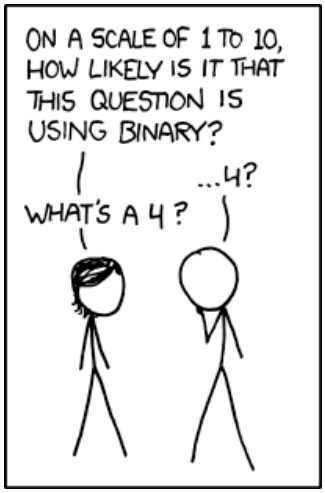
Δυαδικοί τελεστές και μάσκες

Το παρόν φυλλάδιο συνοδεύει τη διάλεξη σχετικά με δυαδικούς τελεστές και μάσκες. Προϋποθέτει πως έχετε ήδη παρακολουθήσει τη διάλεξη κι έχετε εξοικειωθεί με τις βασικές έννοιες που παρουσιάστηκαν σε αυτή. Περιλαμβάνει πρόσθετες επεξηγήσεις και παραδείγματα, με στόχο την ενίσχυση της κατανόησης.



Πηγή: xkcd.com

Γιατί χρησιμοποιούμε δυαδικούς τελεστές

Όπως γνωρίζετε από το πρώτο μάθημα, όλα τα δεδομένα αποθηκεύονται σε έναν υπολογιστή ως ακολουθίες από 0 και 1 δηλαδή ως ακολουθίες από bits. Οι δυαδικοί τελεστές μας επιτρέπουν να επεξεργαζόμαστε απευθείας αυτά τα bits. Γιατί είναι χρήσιμη αυτή η δυνατότητα;

➢ Μας επιτρέπουν να αποθηκεύουμε πολλές μικρές πληροφορίες σε λίγο χώρο.

Ένα παράδειγμα εμφανίζεται στο Linux: τα δικαιώματα πρόσβασης αρχείων αποθηκεύονται ως 9 bits οργανωμένα σε τρεις τριάδες - μία για τον κάτοχο του αρχείου, μία για την ομάδα στην οποία ανήκει (π.χ. πρωτοετείς) και μία για όλους τους χρήστες του συστήματος.

Το λιγότερο σημαντικό bit κάθε τριάδας αφορά την άδεια εκτέλεσης (x), το δεύτερο την άδεια εγγραφής (w) και το τρίτο την άδεια ανάγνωσης (r). Για παράδειγμα, το 111101000 σημαίνει ότι:

- ο ιδιοκτήτης μπορεί να διαβάσει, γράψει και εκτελέσει το αρχείο,
- οι χρήστες που είναι στην ίδια ομάδα με αυτόν να το διαβάσουν και να το εκτελέσουν,
- οι υπόλοιποι χρήστες δεν έχουν πρόσβαση.

Μπορείτε να δείτε αυτές τις πληροφορίες μέσα από ένα τερματικό, με διαφορά ότι αντί για 1 ή 0 εμφανίζονται τα γράμματα r, w, x (αν η άδεια υπάρχει) ή παύλα - (αν η άδεια δεν υπάρχει). Για παράδειγμα, οι παραπάνω άδειες εμφανίζονται ως `rwxr-x---`. Ανοίξτε ένα τερματικό σε έναν κατάλογο που έχετε τα αρχεία ενός εργαστηρίου και γράψτε την εντολή `ls -l` (ελ ες παύλα ελ) για να δείτε τα δικαιώματα πρόσβασης των αρχείων σας!

Ένα άλλο παράδειγμα συναντάται στα παιχνίδια: οι πληροφορίες για την κατάσταση κίνησης ενός χαρακτήρα μπορούν να αποθηκευτούν σε ένα byte, όπου κάθε bit υποδηλώνει αν μια συγκεκριμένη κατάσταση (π.χ. περπάτημα, τρέξιμο, άλμα, σκύψιμο κτλ.) είναι ενεργή (1) ή όχι (0).

- Μας επιτρέπουν να αλλάζουμε ή να εξετάζουμε μεμονωμένα bits

Όταν αποθηκεύουμε πολλές πληροφορίες σε ένα ή περισσότερα byte όπως στα παραπάνω παραδείγματα, χρειαζόμαστε ένα τρόπο να δουλέψουμε με μεμονωμένα bits.

- Οι πράξεις με δυαδικούς τελεστές είναι ταχύτερες από τις αντίστοιχες αριθμητικές

Αυτό είναι ιδιαίτερα σημαντικό σε συστήματα όπου ο χρόνος εκτέλεσης είναι κρίσιμος, όπως μικροελεγκτές, ενσωματωμένα συστήματα και άλλο χαμηλού επιπέδου λογισμικό.

Μια συχνή πράξη είναι ο πολλαπλασιασμός με δύναμη του 2 μέσω αριστερής ολίσθησης (`x << n`), ο οποίος είναι σημαντικά πιο "φτηνός" υπολογισμός από τον `x * pow(2,n)`. Η pow είναι συνάρτηση της μαθηματικής βιβλιοθήκης της C που υπολογίζει ύψωση σε δύναμη.

Εκτός από πράξεις, οι δυαδικοί τελεστές μας επιτρέπουν να κάνουμε γρήγορους ελέγχους όπως:

- Αν ένας ακέραιος είναι ζυγός ή μονός (αν το `x&1` είναι 1, το `x` είναι ζυγός, διαφορετικά μονός),
- Αν ένας θετικός ακέραιος είναι δύναμη του δύο (ισχύει αν και μόνο αν το `x&(x-1)` είναι 0)

Σημειώστε πως αυτές οι τεχνικές συναντώνται πιο σπάνια σε "συμβατικά" προγράμματα γιατί θεωρούνται λιγότερο ευανάγνωστες.

Τι είναι οι μάσκες και σε τι χρησιμεύουν

Μία μάσκα είναι ένας αριθμός που χρησιμοποιείται για να επιλέξουμε, να ελέγξουμε ή να τροποποιήσουμε συγκεκριμένα bits ενός άλλου αριθμού μέσω μιας πράξης με δυαδικούς τελεστές. Η μάσκα **δεν αλλάζει** τον αρχικό αριθμό, αλλά παράγει ένα νέο αποτέλεσμα.

Μια μάσκα τυπικά έχει:

- όλα τα bits 0 εκτός από αυτά που μας ενδιαφέρουν, ή
- όλα τα bits 1 εκτός από αυτά που μας ενδιαφέρουν

Ανάλογα με το τι θέλουμε να πετύχουμε, επιλέγουμε κατάλληλη μάσκα και την κατάλληλη πράξη (AND, OR ή XOR) ανάμεσα σε αυτή και στον αρχικό αριθμό.

Έλεγχος της τιμής ενός bit (testing a bit)

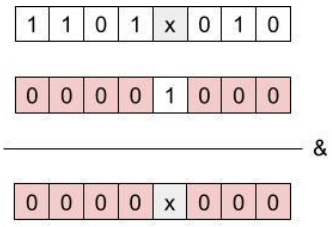
Ας υποθέσουμε πως θέλουμε να ελέγξουμε τι τιμή έχει το 4ο λιγότερο σημαντικό bit ενός αριθμού.

Για να απομονώσουμε το 4ο bit, θέλουμε να δημιουργήσουμε έναν αριθμό που:

α) Στην 4η (λιγότερο σημαντική) θέση έχει 1 ανεξαρτήτως του τι bit υπάρχει στην αντίστοιχη θέση του αρχικού αριθμού, και
β) Σε όλες τις άλλες θέσεις έχει 0 ανεξαρτήτως του τι bit υπάρχει στις αντίστοιχες θέσεις του αρχικού αριθμού.

Για να επιτευχθεί το (α) πρέπει να γίνει η πράξη AND ανάμεσα στο bit της 4ης θέσης και στο 1, γιατί αυτή θα έχει πάντα ως αποτέλεσμα την τιμή του bit της 4ης θέσης.

Για να επιτευχθεί το (β) πρέπει να γίνει η πράξη AND ανάμεσα σε κάθε ένα bit (εκτός του 4ου) και στο 0, γιατί αυτή θα έχει πάντα αποτέλεσμα 0.



Ας εξετάσουμε την τιμή x του 4ου λιγότερο σημαντικού bit του αριθμού 1101x010.

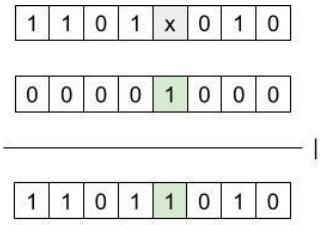
Η επιλογή της μάσκας 0001000 (1 στη θέση που μας ενδιαφέρει και 0 στις άλλες) σε συνδυασμό με την πράξη & μας δίνει το αποτέλεσμα 0000x000. Αν αυτό είναι ίσο με 0 σημαίνει πως το x είναι 0, διαφορετικά το x είναι 1.

➢ Πώς κατασκευάζουμε τη μάσκα;

Η μάσκα κατασκευάζεται ολισθαίνοντας τον αριθμό 1 όσες θέσεις αριστερά χρειάζεται ώστε να βρεθεί στην ίδια θέση με το bit που μας ενδιαφέρει, δηλαδή στην 4η θέση στο συγκεκριμένο παράδειγμα. `mask = 1 << 3; // 3` γιατί το μέτρημα ξεκινά από 0

Αλλαγή της τιμής ενός bit σε 1 (setting a bit)

Ας υποθέσουμε ότι θέλουμε να κατασκευάσουμε ένα νέο αριθμό ο οποίος είναι ίδιος με τον αρχικό, αλλά έχοντας θέσει το 4ο bit σε 1 ανεξαρτήτως του τι τιμή είχε αρχικά. Τα υπόλοιπα bits παραμένουν ίδια.



Στην ίδια λογική με πριν, διαπιστώνουμε πως πρέπει να γίνει η πράξη OR ανάμεσα στον αρχικό αριθμό και σε μία μάσκα που έχει 1 στην 4η θέση και 0 στις υπόλοιπες.

Αλλαγή της τιμής ενός bit σε 0 (clearing a bit)

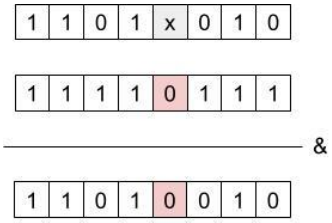
Ας υποθέσουμε ότι θέλουμε να κατασκευάσουμε ένα νέο αριθμό ο οποίος είναι ίδιος με τον αρχικό, αλλά έχοντας θέσει το 4ο bit σε 0 ανεξαρτήτως του τι τιμή είχε αρχικά. Τα υπόλοιπα bits παραμένουν ίδια.

Αυτή τη φορά, ο αριθμός που θα κατασκευάσουμε:

α) Στην 4η (λιγότερο σημαντική) θέση έχει 0 ανεξαρτήτως του τι bit υπάρχει στην αντίστοιχη θέση του αρχικού αριθμού, και
β) Στις άλλες θέσεις έχει τα bit που υπάρχουν στις αντίστοιχες θέσεις του αρχικού αριθμού.

Για να επιτευχθεί το (α) πρέπει να γίνει η πράξη AND ανάμεσα στο bit της 4ης θέσης και στο 0, γιατί αυτή θα έχει πάντα ως αποτέλεσμα την τιμή του bit της 4ης θέσης.

Για να επιτευχθεί το (β) πρέπει να γίνει η πράξη AND ανάμεσα σε κάθε ένα bit (εκτός του 4ου) και στο 1, γιατί αυτή θα έχει πάντα ως αποτέλεσμα την τιμή του bit, ανεξαρτήτως αν είναι 0 ή 1.



Αυτή τη φορά ο στόχος επιτυγχάνεται με πράξη AND ανάμεσα στον αρχικό αριθμό και μία μάσκα που έχει παντού 1 εκτός από την 4η θέση.

➢ Πώς κατασκευάζουμε τη μάσκα;

Παρατηρούμε πως αυτή η μάσκα είναι το συμπλήρωμα της μάσκας που κατασκευάσαμε νωρίτερα:

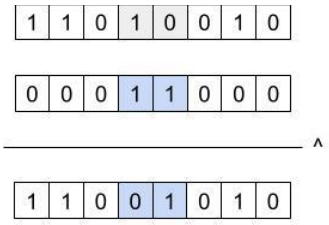
```
mask = ~(1 << 3);
```

Αναστροφή ενός bit (flipping/toggling a bit)

Ας υποθέσουμε ότι θέλουμε να κατασκευάσουμε ένα νέο αριθμό ο οποίος είναι ίδιος με τον 11010010, αλλά έχοντας αλλάξει το 4ο bit και το 5ο bit στις αντίστροφες τιμές τους: αν είναι 0 σε 1, αν είναι 1 σε 0.Τα υπόλοιπα bits παραμένουν ίδια.

Εφόσον θέλουμε αναστροφή, ο πιο κατάλληλος τελεστής είναι το XOR. Παρατηρούμε πως:

- Αν ένα bit είναι 1, τότε η πράξη 1 XOR 1 δίνει 0.
- Αν ένα bit είναι 0, τότε η πράξη 0 XOR 1 δίνει 1.
- Για οποιοδήποτε bit x, η πράξη x XOR 0 δίνει x.



Αυτή τη φορά ο στόχος επιτυγχάνεται με πράξη XOR ανάμεσα στον αρχικό αριθμό και μία μάσκα που έχει 1 στις θέσεις που μας ενδιαφέρουν και 0 στις άλλες.

➢ Πώς κατασκευάζουμε τη μάσκα;

Γνωρίζουμε πως η έκφραση `1 << 3` παράγει μια μάσκα που έχει ενεργό μόνο το bit της 4ης θέσης και η έκφραση `1 << 4` παράγει μια μάσκα που έχει ενεργό μόνο το bit της 5ης θέσης. Η σύζευξή τους παράγει ακριβώς τη μάσκα που χρειαζόμαστε:

```
mask = (1 << 3) | (1 << 4)
```

Σύνθετες μάσκες

Όπως είδατε στο προηγούμενο παράδειγμα, συχνά χρειαζόμαστε μια μάσκα που δρα σε περισσότερα από ένα bits. Ένας τρόπος να κατασκευαστεί μια τέτοια μάσκα παρουσιάστηκε παραπάνω.

Αν χρειαζόμαστε μια πιο σύνθετη μάσκα, τυπικά την αρχικοποιούμε απευθείας σε έναν αριθμό εκφρασμένο στο δεκαεξαδικό σύστημα

➢ Γιατί χρησιμοποιούμε το δεκαεξαδικό σύστημα;

- Γιατί η μετατροπή ανάμεσα στο δεκαεξαδικό και στο δυαδικό σύστημα είναι πολύ πιο εύκολη από τη μετατροπή ανάμεσα στο δεκαδικό και στο δυαδικό σύστημα.
- Γιατί οι αριθμοί στο δεκαεξαδικό σύστημα είναι πιο σύντομοι - είναι πιο εύκολο να γράψουμε 1C5A παρά 1110001111010

Δεκαεξαδικό ↔ Δυαδικό

Κάθε ψηφίο του δεκαεξαδικού συστήματος (hex digit) αντιστοιχεί ακριβώς σε 4 bits:

Hex:	0	1	2	3	4	5	6	7
Bin:	0000	0001	0010	0011	0100	0101	0110	0111

Hex:	8	9	A	B	C	D	E	F
Bin:	1000	1001	1010	1011	1100	1101	1110	1111

➢ Από δεκαεξαδικό σε δυαδικό

Αντικαθιστούμε κάθε hex digit με τα αντίστοιχα 4 bits

➢ Από δυαδικό σε δεκαεξαδικό

Χωρίζουμε τα bits σε ομάδες των 4 ξεκινώντας από δεξιά (συμπληρώνοντας με μηδενικά αριστερά αν χρειάζεται) και αντικαθιστούμε κάθε ομάδα με το αντίστοιχο hex digit.

Παράδειγμα:

```
1110001111010 -> 0001 1100 0111 1010 -> 1C7A
```

Χρήση σύνθετης μάσκας

Ας υποθέσουμε ότι θέλουμε να αντιστρέψουμε κάθε δεύτερο bit ενός αριθμού μεγέθους 4 bytes (32 bits), ξεκινώντας από το λιγότερο σημαντικό. Χρειαζόμαστε μια μάσκα που στο δυαδικό σύστημα έχει τη μορφή `101010...1010`. Στο δεκαεξαδικό σύστημα η μάσκα γράφεται ως `0xAAAAAAAA` .

Άρα, η πράξη που αντιστρέφει τα bits είναι: `number = number ^ 0xAAAAAAAA;`

Εφαρμογή

Εάν έχετε ασχοληθεί με κατασκευή ιστοσελίδων, θα έχετε παρατηρήσει πως τα χρώματα εκφράζονται στο δεκαεξαδικό σύστημα, για παράδειγμα, 0xD1CD8E. Αυτό είναι ένα παράδειγμα της RGB μορφής.

Κάθε χρώμα είναι ένας συνδυασμός από:
κόκκινο (Red),
πράσινο (Green) και
μπλέ (Blue)

Bits	Περιεχόμενο	Παράδειγμα
16-24	Red	D1
8-15	Green	CD
0-7	Blue	8E

Το λευκό χρώμα είναι `0xFFFFFF` ενώ το μαύρο `0x000000`.

Σε κάποιες εφαρμογές χρησιμοποιείται και το 4ο byte για να αναπαραστήσει τη διαφάνεια (alpha). Η αναπαράσταση τότε λέγεται RGBA ή ARGB ανάλογα με το αν το alpha είναι στο λιγότερο ή περισσότερο σημαντικό byte.

➢ Παιχνίδια με χρώματα

Γράψτε ένα πρόγραμμα που ορίζει μια μεταβλητή τύπου unsigned int στην οποία θα αποθηκευτεί ένα χρώμα.

Ζητήστε από το χρήστη να εισάγει ένα αρχικό χρώμα από το πληκτρολόγιο (σε δεκαεξαδική μορφή) και στη συνέχεια δώστε μια σειρά από επιλογές (ευκαιρία για χρήση if ή switch). Για παράδειγμα:

- Αφαίρεση ενός εκ των R, G, B (μηδενισμός του αντίστοιχου byte)
- Reset ενός εκ των R, G, B (αλλαγή του αντίστοιχου byte σε `0xFF`)
- Αλλαγή ενός εκ των R, G, B (σκεφτείτε πώς θα γίνει αυτό - υπάρχει ήδη μια τιμή γι' αυτό το χρώμα και θέλουμε να έχει μια διαφορετική. Hint: η αλλαγή θα γίνει σε δύο βήματα)

Χρησιμοποιήστε ένα site όπως το https://rgbacolorpicker.com/hex-to-rgba για να δείτε τα χρώματα που κατασκευάζετε.

Στη συνέχεια, ζητήστε από το χρήστη να σας δώσει ξεχωριστά τις τιμές και των τριών χρωμάτων και γράψτε κώδικα που κατασκευάζει την τελική τιμή του χρώματος. Για παράδειγμα, αν ο χρήστης δώσει `0x35` για το red, `0xBC` για το green και `0xC4` για το blue, το πρόγραμμα σας πρέπει να κατασκευάζει τον αριθμό `0x35BCC4`.

Προαιρετικά, μπορείτε να πειραματιστείτε και με τη διαφάνεια χρησιμοποιώντας τη μορφή RGBA. Σε αυτή την περίπτωση, το παραπάνω χρώμα γράφεται `0x35BCC4FF` αν θέλετε να είναι αδιαφανές, `0x35BCC47F` αν θέλετε να είναι ημιδιαφανές, ή `0x35BCC400` αν θέλετε να είναι πλήρως διαφανές (οπότε και θα δείτε το λευκό υπόβαθρο στην εικόνα).