Architecture et Système

Stefan Schwoon

Cours L3, 2016/17, ENS Cachan

Entiers

Stockés dans un mot de taille fixe (typiquement 8, 16, 32 bits)

Types en C: `char`, `short int`, `int`, `long int`, `long long int`

Les tailles de ces types ne sont pas précisement définis par le langage de C, seulement leurs tailles minmales : 8, 16, 16, 32, 32, où `int` est un mot de registre.

On peut obtenir les valeurs concrètes avec `sizeof(char)` etc.

Big vs little endian

Un mot de taille > 8 nécéssite plusieurs octets en mémoire.

Big-endian: on stocke le mot le plus significatif dans le premier octet
p.ex. $(12345678)_{16}$ est stocké dans quatre octets dans l'ordrer 12, 34, 56, 78 (hexadécimal)

Little-endian: c'est l'inverse, on stocke les octets dans l'ordre 78, 56, 34, 12

Le mode de stockage devient important quand on échange des données binaires (fichiers, réseau).

Dans ces cas, l'ordre doit être spécifié par le protocol / format de fichier.

P.ex., l'Internet protocol (IP) définit cet ordre comme big-endian.

Fonctions en C : `ntohl`, `ntohs`, `htonl`, `htons`

Entiers avec/sans signe

Un mot de n octets peut être stocke les valeurs $0 \ldots 2^n - 1$.
On l'appele un entire sans signe (*unsigned*). Les opération arithmétiques travaillent implicitement modulo 2^n.

Un entier avec signe (*signed*) est typiquement stocké en format complément à deux où un mot stocke des valeurs $-2^{n-1} \ldots 2^{n-1} - 1$.

Pour les valeurs non-negative, le bit le plus significatif (MSB) est de 0.

Pour les valeurs négatives, le MSB est de 1: représentation de $-i$ (pour $i \geq 0$) obtenue en soustrayant 1, puis en prenant la négation (bit par bit) du résultat.

P.ex., -1 représenté par $11 \cdots 1$, -2^{n-1} par $10 \cdots 0$.

Du coup, l'addition de i and $-i$ en utilisant l'addition *sans signe* donne `0`, le bon résultat.

Conclusion : sur le niveaux binaire, les opérations arithmétiques pour les naturels (p.ex. addition/multiplication) donnent aussi le bon résultat pour les entiers avec signe.

$\Longrightarrow$ la distinction sans/avec signe n'existe pas pour le processeur – c'est simplement une façon d'interpréter (lors de la saisie/affichage sur l'écran) !

En C

Les types entiers peuvent être déclaré comme `signed` ou `unsigned` ; par défaut ils sont `signed`.

Pour des `char` cette distinction n'est pas important lorsqu'on les interprète comme des caractères.

Attention aux opérateurs de décalage (<< et >>) :

pour les `unsigned`, l'opération est dite *logique*, le décalage se fait sur l'intégralité du mot, en inclus le bit le plus significatif;

pour les `signed`, l'opération est dite *arithmétique*, elle conserve le signe.

Valeurs réelles

Valeurs réelles représentées typiquement dans un format virgule flottante (mots de taille fixe).

Du coup : précision limité

Idée en général : tuple (s, m, e) avec l'interprétation $\pm 2^e \cdot m$.

- s est le signe (un seul bit, 0 non-négatif, 1 négatif);
- m est la mantisse;
- e est l'exposant.

Permet un compromis entre taille et précision des valeurs représentables.

Besoin des standards

Problèmes:

taille d'exposant, mantisse?

représentations pas uniques : $(s, m, e) \equiv (s, 2m, e - 1)$

Comment traiter des cas spéciaux (division par zéro), comment traiter les arrondis ?

Le standard le plus important s'appele IEEE 754.

En C : `float` en C = IEEE 754, 32 bit; `double` = IEEE 754 (64-bit).

Dans le suivant, on discutera la partie 32 bit, les autres parties étant similaires.

IEEE 754 (variante 32-bit)

IEEE 754 spécifie les conventions suivantes :

1 bit pour le signe, 8 pour l'exposant, 23 pour la mantisse;

Interpretation d'exposant: $e_u - 127$, où e_u est l'interprétation *sans signe* des 8 bits. Du coup, on représente ± 127. La valeur 128 est reservée pour quelques cas spéciaux.

Interprétation de la mantisse: $1 + (m_u/2^{23})$, ce qui donne $[1, 2)$.

Remarques :

Interprétation de la mantisse force une représentation unique.

$e = 128$ pour $\pm\infty$ (avec $m_u = 0$)
ou NaN (not a number, avec $m_u \neq 0$)

Addition dans IEEE

Le standard IEEE définit la procédure à suivre pour effectuer des opérations arithmétiques (comment arrondir, comment traiter des cas spéciaux, . . .).

Discutons le cas d'une addition:

Soient $x = 2^{e_x} \cdot m_x$ et $y = 2^{e_y} \cdot m_y$ et $x > y$
(par simplicité on suppose qu'ils sont tous les deux positifs).

P.ex. $x = 2.5$ et $y = 0.75$, du coup,
$e_x = 1, m_x = 1.25, e_y = -1, m_y = 1.5$.

D'abord, on représente les mantisses en tant qu'entier, en prenant compte de la "'une cachée"). Ceci donne deux entiers $i_x = 0\text{xa}00000$ et $i_y = 0\text{xc}00000$ (pour 23 bits de mantisse).

Ensuite on adapte i_y par rapport à la différence des exposants (décalage par $1-(-1)=2$ au droite), i_y devient $0x300000$.

L'addition entière entre i_x et i_y donne $i_z = 0xd00000$.

Comme le résultat n'excède pas les 24 bits (une cachée plus 23 bits), on garde l'exposant de x et on enlève simplement l'une cachée de i_z pour obtenir la mantisse du résultat.

Si i_z avait débordé les 24 bits, il aurait d'abord fallu décaler i_z à droit par une position (perdant un bit de précision).

Problèmes de virgule flottante

Précision limitée : certaines valeurs “rondes” comme 0.1 ou 2.3 ne sont pas représentables avec exactitude.

Erreurs d’arrondi : $x + y$ donne x si x beaucoup plus grand qu’y.

Du coup, certaines lois comme distributivité ne sont plus valables.

Codage et traitement des caractères

Sujet un peu difficile car :

beaucoup de standards différents et partiellement compatibles

support souvent insuffisant dans les langages de programmation

ignorance des programmeurs/utilisateurs/institutions

représentation en texte cache sa représentation, difficile à debugger

Resumé du suivant et du TP

Leçons à retenir :

Un caractère n'égale pas (toujours) à un octet

Pour traiter du texte il faut connaître son codage de caractères.
(et cette information manque souvent !)

C'est un enjeu socio-culturel.

Survol et historique des standards existants

Traitement dans quelques applications importantes (terminal, LaTeX, web, mail)

Reperer (et corriger) les erreurs les plus souvent rencontrées

Terminologie

Pour connaître les différents standards de codage de caractères, il sera utile de distinguer les termes suivants :

jeu de caractères : l'ensemble (non-ordonné) des caractères (graphèmes) dont on dispose
On note $A \subseteq B$ si le jeu du standard A est un sous-ensemble de celui de B.

jeu de caractères codés : fonction qui affecte à chaque graphème un *code* numérique.
On note $A \prec B$ si $A \subset B$ et A affecte tous ses caractères aux mêmes codes que B.

codets : unité la plus petite utilisée pour un codage donné.
Un *code* peut être codé par une suite de plusieurs codets.

forme de codage : comment représenter un codet

Historique : Les débuts

Premiers ordinateurs : aucun traitement de texte, on fait des calculs !

Années 60 : ordinateurs partagés, utilisateurs y accédent à l'aide des terminaux :

Appareils simple sans aucune "intelligence" propre

Affichent les caractères saisis par l'utilisateur (si terminal en mode "echo") + transfert vers l'ordinateur

Affichent les données venues de l'ordinateur (caractères imprimables et de contrôle)

Caractères de contrôle: retour chariot (CR), saut de ligne (LF), appel (BEL), retour arrière (BS), ...

Parenthèse : nouvelle ligne

Grâce à des raisons historiques, il existe deux standards pour représenter une nouvelle ligne :

format Unix : un seul caractère (10 décimal) entre deux lignes

format DOS: nouvelle ligne représentée par CR+LF = caractères 13+10

La plupart des éditeurs de texte savent traiter les deux formats.

Conversion explicite : `dos2unix`, `unix2dos`

En `vi`: option `fileformat` (=unix,dos)

Les premiers standards

EBCDIC (créé par IBM (largement insignifiant aujourd'hui)

International : ISO-646 in 1963/1972:

code de 7 bit, avec quelques positions laissées à la disposition des autorités nationales

ASCII : variante US d'ISO-646, devient le standard *de facto*

La suite: que faire du huitième bit ?

IBM PC (1981) : diverse “pages de code”, selon les pays
(dans les payes occidentaux: codepage 437)

ISO-8859 (à partir de 1985) : diverse variants, le plus important étant
ISO-8859-1 (= Latin-1), avec encore quelques positions inutilisées

Windows-1252 (arrivée avec MS Windows) : extension de Latin-1

$$\text{ASCII} \prec \text{Latin-1} \prec \text{Windows-1252}$$

Unicode/ISO 10646

(à partir de 1991) effort pour définir un codage *universelle*, c'est à dire pour pouvoir représenter tous les langages du monde.

version actuelle connaît 1 million de caractères différents

Latin-1 $\prec$ Unicode et Windows-1252 $\subseteq$ Unicode

Usage actuel :

plus de 80% des pages web modiales en Unicode, 10% en Latin-1

Codage de Latin-1 et Unicode

Latin-1 : C'est facile, un octet égale un caractère !

Unicode : Plusieurs codages différents, le plus important étant UTF-8 :

Un code (=caractère) représenté par un ou plusieurs *codets*.

Pour les caractères ASCII: un seul codets entre 0..127.

Pour les autres : 2 à 4 codets entre 127..255.

Un codet représenté par un octet ou dans d'autres formes (p.ex., quoted-printable)

Traitement dans le terminal

Aujourd'hui : Le terminal est une application X qui simule le comportement d'un appareil ancien.

Le codage de caractère peut être configurée dans les options du terminal.

Le terminal reçoit les clés tapès par l'utilisateur et les affiche sur l'écran (si en mode echo) + transfert vers le processus du premier plan (p.ex. le shell) dans le bon codage.

Parenthèse

Qu'est-ce qui se passe lorsqu'on tape (ou relâche) un clé ?

Le clavier emet une interruption au processeur.

Le processeur appelle le gestionnaire d'interruption pour le clavier.

Ceci (fourni par le système d'exploitation) obtient le code de clé.

Le système traduit le code de clé vers un code de symbole, selon la configuration de l'utilisateur.

Sous X11, le systeme transmet un "evènement" à l'application concernée.

Traitement dans TeX

TeX/LaTeX : né dans l'ère 7-bit/ASCII

fait son propre traitement de caractères :

les accents et les lettres sont des graphèmes separés

n'importe quel accent peut être utilisé avec n'importe quelle lettre

package `inputenc` permet de saisir des caractères Latin-1/UTF-8 etc.

Traitement dans les navigateurs Web

Plusieurs façons de spécifier les codages, il existe un standard détaillé pour décider lequel un navigateur doit utiliser :

dans les meta-données du fichier HTML

dans les entêtes du protocol HTTP

détection automatique (heuristique)

...

Traitement dans les mails

Un mail consiste des entêtes et du texte proprement dit.

Tous les entêtes doivent être en ASCII.

Les entêtes doivent définir le codage utilisé dans le mail.

Pour utiliser des caractères non-ASCII, il existe des mots MIME.