Architecture des ordinateurs

Fiche n°1

ESIPE - INFO 1 2024 –2025

Quelques bases de représentation de la donnée

Le but de ce TP est d'introduire des premières bases qui serviront à la programmation en assembleur sur des processeurs 32 bits d'architecture x86, sous Linux, et en mode protégé.

Exercice 1 (Nombre de données)

1. Dessiner un tableau à deux lignes dont la ligne du haut est indexée par les entiers allants de 0 jusqu'à 16 et la ligne du bas contient les valeurs 2^0 jusqu'à 2^{16}. Apprendre ce tableau par cœur.
2. Combien de données différentes peut-on coder sur n bits ?
3. Combien de données différentes peut-on coder sur n octets ?
4. Combien de données différentes peut-on coder sur n Kio ?
5. Combien de données différentes peut-on coder sur n Mio ?
6. Combien de données différentes peut-on coder sur n Gio ?

---

1 Représentation des entiers

Exercice 2 (Changements de base)

1. Représenter la valeur $(999)_{\text{dix}}$ en base deux, en base quatre et enfin en base seize. L'exécution de l'algorithme de changement de base doit être illustrée pas à pas.

   Note : en base 16, on utilise les lettre `abcdef` *pour compléter les 10 chiffres usuels.*

2. Représenter la valeur $(1002)_{\text{trois}}$ en base dix.

Exercice 3 (Binaire et hexadécimal)

1. Représenter la valeur $(\texttt{10101011111000000000001})_{\text{deux}}$ en hexadécimal en utilisant la méthode de conversion rapide traduisant chaque suite de quatre bits en une lettre hexadécimale.
2. Représenter la valeur $(\texttt{2BABA101F9})_{\text{seize}}$ en binaire en utilisant la méthode de conversion rapide traduisant chaque lettre hexadécimale en une suite de quatre bits.

Exercice 4 (Représentation avec biais)

1. Représenter sur $n := 8$ bits avec un biais de $k := 1000$ la valeur $(-950)_{\text{dix}}$ en représentation avec biais.
2. Représenter sur $n = 16$ bits avec un biais de $k := 256$ la valeur $(0)_{\text{dix}}$ en représentation avec biais.
3. Donner la valeur décimale représentée par la suite de bits $(\texttt{00000000})_{\text{biais=21}}$.
4. Donner la valeur décimale représentée par la suite de bits $(\texttt{0000000000010001})_{\text{biais=17}}$.
5. Donner la valeur décimale représentée par la suite de bits $(\texttt{00010001})_{\text{biais=1777}}$.

Exercice 5 (Représentation en complément à deux)

1. Représenter sur $n := 8$ bits la valeur $(17)_{\text{dix}}$ selon la représentation en complément à deux.
2. Représenter sur $n := 8$ bits la valeur $(-1)_{\text{dix}}$ selon la représentation en complément à deux.
3. Représenter sur $n := 16$ bits la valeur $(-1021)_{\text{dix}}$ selon la représentation en complément à deux.
4. Expliquer s'il est possible de représenter la valeur $(-1000)_{\text{dix}}$ en complément à deux sur $n := 8$.
5. Donner la valeur décimale représentée par la suite de bits $(\texttt{00001001})_{\text{c2}}$ sur $n := 8$ bits.
6. Donner la valeur décimale représentée par la suite de bits $(\texttt{10001001})_{\text{c2}}$ sur $n := 8$ bits.
7. Donner la valeur décimale représentée par la suite de bits $(\texttt{11001011})_{\text{c2}}$ sur $n := 8$ bits.

1.1 Opérations en complément à deux

On rappelle que la représentation des entiers *signés* se fait sur un nombre de bits fixé. Dans ce cas, l'addition de deux entiers peut produire une retenue dite *de sortie* qui ne fait pas partie de la représentation du résultat.

Par exemple, pour $n := 4bits$, `1110 + 0101 = (1)0011` on obtient une retenue de sortie.

Aussi, le résultat de l'addition peut ne pas être représentable sur le même nombre de bits, dans ce cas il y a dépassement de capacité.

Par exemple pour $n := 4bits$, la valeur -5 peut être représenté en $(\texttt{1011})_{\text{c2}}$ mais la somme $-5 + -5 = -10$ ne rentre pas dans la plage de valeur possible sur 4 bits.

Exercice 6 (Critère de dépassement de capacité)

Proposer une méthode pour détecter sur le calcul en binaire les cas de dépassement de capacité.

Exercice 7 (Dépassement de capacité et retenue de sortie)

Considérons les valeurs suivantes sur $n := 8$ bits :

- $v_1 := (\texttt{10011100})_{c2}$
- $v_2 := (\texttt{00000101})_{c2}$
- $v_3 := (\texttt{10001000})_{c2}$
- $v_4 := (\texttt{11111001})_{c2}$
- $v_5 := (\texttt{00110101})_{c2}$
- $v_6 := (\texttt{11100000})_{c2}$
- $v_7 := (\texttt{01001111})_{c2}$
- $v_8 := (\texttt{01011001})_{c2}$
- $v_9 := (\texttt{10000000})_{c2}$
- $v_{10} := (\texttt{11000000})_{c2}$

Réaliser, en les posant, les additions suivantes et pour chacune d'elle, dire s'il y a une retenue sortante et/ou un dépassement de capacité.

1. $v_1 + v_2$;
2. $v_3 + v_4$;
3. $v_5 + v_6$;
4. $v_7 + v_8$;
5. $v_9 + v_{10}$;
6. $v_1 + v_{10}$.

Les additions posées font partie de la réponse et doivent être utilisées pour justifier les résultats fournis.

---

2 Boutisme et mémoire

La mémoire est organisée par octet plutôt que par bit. Pour stocker un *double word* il faut 4 octets donc 4 cases dans la mémoire. L'ordre de ces octets dans la mémoire dépend de l'architecture de la machine utilisée, de l'octet de poids fort à l'octet de poids faible (grand boutisme) ou de l'octet de poids faible à l'octet de poids fort (petit boutisme).

Les architectures qui nous intéressent sont en petit boutisme.

Exercice 8 Aux adresses 8 à 11 de la mémoire, on trouve les octets $(\texttt{49})_{\text{seize}}$, $(\texttt{31})_{\text{seize}}$, $(\texttt{61})_{\text{seize}}$ et $(\texttt{4f})_{\text{seize}}$.

Donner la représentation hexadécimale du *double word* à l'adresse 8.

---

3 Codage du texte

Pour représenter du texte, chaque lettre/symbole est représenté par un nombre entier lui même représenté en octets (suivant différents encodages). Une suite de lettre devient donc une suite de nombre (puis d'octets).

Les tables de correspondance entre symboles et nombres sont dictés par des normes dont une des plus utilisés est la norme ASCII datant des années 1960.

Chaque lettre est représentée par un entier lui même représentatble sur 7 bits (cf figure suivante) (d'un même octet).

Exercice 9 (Un peu d'ASCII)

1. Représenter le texte $(\texttt{Bien})_{\text{ASCII}}$ en représentation héxadécimale sur 4 octets.
2. Représenter les lettres `'d'`, `'D'`, `'t'` et `'T'` en représentation binaire sur $n := 8$ bits. Qu'observe-t-on ?
3. En observant la représentation hexadécimal des chiffres de la table ASCII, donner une astuce pour retrouver rapidement l'un à partir de l'autre.

Bits b_4	b_3	b_2	b_1	Row / Column	0 (000)	1 (001)	2 (010)	3 (011)	4 (100)	5 (101)	6 (110)	7 (111)
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	¦
1	1	0	1	13	CR	GS	—	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	_	o	DEL

Figure 1: Table ASCII

4. On trouve dans la mémoire l'entier 695160616 codé sur un double word en petit boutisme. Donner le texte correspondant.

5. Donner la représentation ASCII des octets de l'exercice précédent.

Après dénombrement, on s'aperçoit rapidement qu'il n'est pas possible de représenter tous les symboles possible (à commencer par nos lettres accentuées).

D'autres normes ont vu le jour, en restant rétro-compatible avec la norme ASCII. Une des plus utilisées est la norme Unicode. Par exemple, les lettres "à" et "œ" sont représentées par les nombre $(\texttt{e0})_{\text{seize}}$ et $(\texttt{153})_{\text{seize}}$. La valeur maximale permise par la norme est $(\texttt{10ffff})_{\text{seize}}$.

Pour représenter chacune de ses valeurs, il faut de 1 à 3 octets. Cela introduit une difficulté quant à l'encodage en octets que l'on va illustrer par l'exercice suivant.

Exercice 10 (Suite d'entier)

1. Proposer différentes méthodes pour représenter la suite d'entier `14 0 35 1545 1 1 0` avec uniquement les chiffres (sans espaces donc). Elles doivent fonctionner pour toutes suites de nombres de moins de *9* chiffres.

2. Les comparer.

3. La norme Unicode définit plusieurs encodages UTF. Sont-ils comparables aux méthodes précédentes ?
