Jeu d'instructions: x86
Architecture: 32 bits
OS: Linux
Syntaxe: Intel/NASM
Convention: System V ABI i386

Compilation	
$> nasm -f elf a.asm (-g -F dwarf) $> gcc -o a.exe a.o -m32 (-ggdb) $> objdump -d -M intel -r -S -l --no-show-raw-insn -j .text a.exe	
Sources	http://www.info.univ-angers.fr/~richer/ensl3i.php#ARCHI
	https://www.unilim.fr/pages_perso/tristan.vaccon/cours_nasm.pdf
	https://software.intel.com/sites/default/files/managed/39/c5/325462-sdm-vol-1-2abcd-3abcd.pdf
	https://www.felixcloutier.com/x86/
	https://c9x.me/x86/
	http://info.univ-angers.fr/pub/richer/ens/l3info/ao/253666.pdf
	http://info.univ-angers.fr/pub/richer/ens/l3info/ao/253667.pdf

Partitionnement du programme

.data	contient les instructions: db, dw, dd...
.bss	contient les definitions des variables non-initialisées ->uniquement allouées en mémoire (instructions: resb, resw, resd...)
.text	contient le code executé par le programme

Types de données

Nom	Taille (octets)	Taille (bits)	signification	Notes
BYTE	1	8	Byte	1 char
WORD	2	16	Word	
DWORD	4	32	Double Word	Taille d'un int
QWORD	8	64	Quad Word	
TWORD	10	80	Ten bytes	Taille d'un registre flottant

Pseudo-instructions

Nom	Signification	Description	Taille	Exemple
db	Define Bytes	definit un octet	1 octet	db 'Hello', 10
dw	Define Word	definit deux octets	2 octets	dw 746
dd	Define Double	definit quatre octets	4 octets	dd 0x12345678
dq	Define Quad	definit huit octets non-string	8 octets	
dt	Define Ten	definit 10 octets flottants, non-string	10 octets	dt 1.234567e20
ddq	Define Double Q	definit 8 octets non-flottants, non-string	8 octets	
do	Define Octo	definit 8 octets non-string	8 octets	
resb	Reserve Bytes	declare des octets non initialises	n octets	resb 64
resw	Reserve Word	declare des octets non initialises	n*2 octets	resb 1
resd	Reserve Double	declare des octets non initialises	n*4 octets	
resq	Reserve Quad	declare des octets non initialises	n*8 octets	resq 10
rest	Reserve Ten	declare des octets non initialises	n*10 octets	
resdq	Reserve Double Q	declare des octets non initialises	n*8 octets	
reso	Reserve Octo	declare des octets non initialises	n*8 octets	
incbin	Include Binary	inclut un fichier binaire	petit fichier	incbin 'ex.dat'
equ	Equal	definit une constante	variable	len equ $-msg
times	Times	repete une instruction ou des donnees	variable	times 64 db 0

Instructions

Nom	Signification	Description	Arguments	Exemple
		Transfert des données		
mov	move	transfert de donnees de *source* vers *destination*	dest, src	mov eax, 0
movsx	Move and sign extend	MOV + permet de convertir une valeur de 8 en 16 bits ou de 16 en 32 bits en une **valeur signée**	dest, src	
movzx	Move and zero extend	MOV + permet de convertir une valeur de 8 en 16 bits ou de 16 en 32 bits en une **valeur non signée**	dest, src	movzx eax, byte [esi]
push	push stack	push une valeur sur la stack	value	push rdi
pop	pop stack	pop une valeur de la stack et la stocke dans *reg*	reg	pop rdi
popcnt	population count	compte le nombre de 1	reg	popcnt eax
xchg	exchange	echange le contenu des 2 operandes	a, b	xchg ax, bx
lea	load eff. addr.	charge l'adresse de src dans dest	dest, src	lea rsi, [ebx+8*eax+4]
		Arithmetiques		
add	add	addition	dest, src	add ax, bx
sub	subtract	soustraction	dest, src	sub ebx, 100
mul	multiplication	multiplication non signée	arg	mul ecx

imul	integer multiplication	multiplication signée	arg	imul 2
div	division	division non signée	arg	div 2
idiv	integer division	division signée	arg	idiv 2
adc	add carry	dest + (src+retenue)	dest, src	adc ax, 8
sbb	sub borrow	dest - (src+retenue)	dest, src	sbb ax, 8
inc	increment	incrementation	reg	inc ecx
dec	decrement	decrementation	reg	dec ecx
neg	negate	réalise le complément à 2 (*= -1)	reg	neg eax
Logiques				
and	and	ET	dest, src	and ax, bx
or	or	OU	dest, src	or ax, bx
xor	xor	OU-exclusif	dest, src	xor ax, ax
not	not	réalise le complément	dest	not ax
Decalage de bits				
shl	shift left	<< Equivalent a dest *= 2^nb	dest, nb	shl 1, [ax]
shr	shift right	>> Equivalent a dest /= 2^nb	dest, nb	shr 1, [ax]
sal	shift arithmetic left	shl signe	dest, nb	sal ax, 2
sar	shift arithmetic right	shr signe	dest, nb	sal ax, 2
rol	rotate left	shl faisant rotate les bits qui depassent	dest, nb	rol rax, 4
ror	rotate right	shr faisant rotate les bits qui depassent	dest, nb	ror rax, 4
Control Flow				
jmp	jump	saut à l'adresse adr	adr	jmp begin
je	jump equal	jump si ZF = 1	adr	je _done
jne	jump not equal	jump si ZF = 0	adr	jne _cwhile
jz	jump zero	jump si ZF = 1	adr	jz _done
jnz	jump not zero	jump si ZF = 0 ?	adr	jnz _cwhile
jg	jump greater	jump si SF = OF && ZF = 0	adr	jg _greater
jge	jump greater equal	jump si SF = OF	adr	jge _greatereq
jl	jump less	jump si SF != OF	adr	jl _less
jle	jump less equal	jump si SF != OF || ZF = 1	adr	jle _lesseq
jo	jump overflow	jump si OF = 1	adr	jo _ov
jno	jump not overflow	jump si OF = 0	adr	jno _nov
js	jump signed	jump si SF = 1	adr	js _js
jns	jump not signed	jump si SF = 0	adr	jns _njs
cmp	compare	Compare les valeurs des 2 operandes	a, b	cmp [var], 10
test		Effectue un ET binaire entre les deux valeurs. Met à jour le flag ZF		test ecx, ecx
call	call	appel de sous-programme	sous-programme	call hello
ret	return	retour de sous-programme		ret
syscall	system call	requiert un service au kernel (=int 0x80)		syscall

Multiplication (eax * ...)		
Registre	Bits	Res
AL	* 8 bits	AX
AX	* 16 bits	DX:AX
EAX	* 32 bits	EDX:EAX

Division			
Registre	Bits	Quotient	Reste
AX	/ 8 bits	AL	AH
DX:AX	/ 16 bits	AX	DX
EDX:EAX	/ 32 bits	EAX	EDX

Registres (Intel)

Nom				Signification	Description
64 bits	32 bits	16 bits	8 bits		
Registres generaux					
RAX	EAX	AX	AH / AL	Accumulator	Réalisation d'opérations arithmétiques et logiques
RBX	EBX	BX	BH / BL	Base	Opérande ou registre pointeur
RCX	ECX	CX	CH / CL	Counter	Ccompteur dans les opérations itératives
RDX	EDX	DX	DH / DL	Data	Multiplications & divisions ainsi que l'accès aux circuits d'entrées et sorties
Registres pointeurs & index					
RSP	ESP	SP	SPL	Stack Pointer	Pointeur de pile
RBP	EBP	BP	BPL	Base Pointer	Fait réf. aux paramètres des procédures et fonctions qui sont passés dans la pile
RSI	ESI	SI	SIL	Source Index	Fait référence à la mémoire (cf. MOVSB, LOADSB, SCASB)
RDI	EDI	DI	DIL	Destination Index	Fait référence à la mémoire (cf. MOVSB, STOSB, SCASB)
RIP	EIP			Instr. Pointer	Pointeur sur la prochaine instruction
Registres de segments					
		CS		Code Segment	segment courant du code (CS:EIP contient l'adr next instruction à exec)
		DS		Data Segment	Segment courant des données
		SS		Stack Segment	Segment courant de la pile
		ES		Extra Segments	Segment additionnel
		FS			
		GS			
Registres d'etat (EFLAGS) @TODO ?					

Vectorisation (SEE2 & AVX2)

Registre

Type	Nom	Taille (bits)	Info complementaire
MMX	mm0-7	64	Pas vu en TD !
SSE	xmm0-7	128	
AVX	ymm0-7	256	3 opérandes pour une instruction : dest, src1, src2

						Single Precision	Double Precision
pshufd	xmm2	xmm1	**10 01 00 11**			Float	Double
shufps						32 bits	64 bits
		1 1	**1 0**	**0 1**	**0 0**		
	xmm1	x1	x2	x3	x4		
	xmm2	x2	x3	x4	x1		

Instructions

SSE2

Nom	Signification	Description	Arguments	Exemple
		Traitement des **valeurs entières**		
movd	Move **Doubleword**	r/m32 to xmm xmm register to r/m32	xmm, r/m32 r/m32, xmm	MOVD xmm, r/m32 MOVD r/m32, xmm
movq	Move **Quadword**	r/m64 to xmm xmm register to r/m64	xmm, r/m64 r/m64, xmm	MOVQ xmm, r/m64 MOVQ r/m64, xmm
movdqu	Move unaligned double quadword	mov entiers **non** alignés	xmmDst, xmmSrc/m128 (registre) xmmDst/m128, xmmSrc (mem/reg)	
movdqa	Move aligned double quadword	mov entiers **alignés**		
movss	Move Scalar Single-Precision Floating-Point Values	Fusionne les floats de xmm2 à xmm1 Charge les floats de m32 à xmm1	xmm1, xmm2 xmm1, m32	MOVSS xmm1, xmm2 MOVSS xmm1, m32
paddB	Add packed **Byte**	Add	xmm0, xmm1/m128	PADDB xmm1, xmm2/m128
paddW	Add packed **Word**			PADDW xmm1, xmm2/m128
paddD	Add packed **Doubleword**			PADDD xmm1, xmm2/m128
paddQ	Add packed **Quadword**			PADDQ xmm1, xmm2/m128
pand	Logical AND	DEST = DEST AND SRC		PAND xmm1, xmm2/m128
pcmpeqb	Compare Packed **Bytes** for Equal	Comparaison des octets compactés		PCMPEQB xmm1, xmm2/m128
pcmpeqd	Compare Packed **Words** for Equal			PCMPEQW xmm1, xmm2/m128
pmulld	Multiply Packed Integers and Store Low Result	Fait 4 multiplication 32 bits en parrallèle et stock les 32 bits inferieurs -> dst = dst * src		PMULLD xmm1, xmm2/m128
pmovmsk	Move Byte Mask	mov et les bits superieurs de r32 ou r64 sont à 0	reg, xmm	PMOVMSKB ebx, xmm0
		Traitement des **valeurs réelles**		
movups	Move Unaligned Packed Single	mov flottants **non** alignés	xmmDst, xmmSrc/m128 (registre) xmmDst/m128, xmmSrc (mem/reg)	
movaps	Move Aligned Packed Single	mov flottants **alignés**		
addps	Add Packed **Single Precision**	xmmDst = xmmDst + xmmSrc	xmmDst, xmmSrc/m128	
addpd	Add Packed **Double Precision**			
sqrtps	Square Root of ***Single*-Precision** Floating-Point Values	xmmDst = sqrt(xmmSrc)		
sqrtpd	Square Root of ***Double*-Precision** Floating-Point Values			
minps	Minimum of Packed Single...	retourne le min des 2 registres ds le 1er		MINPS xmm1, xmm2/m128
maxps	Maximum of Packed Single...	retourne le max des 2 registres ds le 1er		MAXPS xmm1, xmm2/m128
xorps	Bitwise Logical XOR...	xor		XORPS xmm1, xmm2/m128
		Conversion et manipulation		
cvtsi2ss	Convert Doubleword Integer to Scalar Single-Precision Floating-Point Value	Converti des entiers en flottants simple précision.	xmmDst, r/m32	CVTSI2SS xmm1, ebx
cvtps2pd	Convert Packed Single-Precision Floating-Point Values to Packed Double-Precision Floating-Point Values	Converti des flottants simple précision en flottants double précision.	xmmDst, xmmSrc	CVTPS2PD xmm1, xmm2/m64
haddps	Packed Single-FP Horizontal Add	xmm1 = x4 | x3 | x2 | x1 xmm2 = y4 | y3 | y2 | y1 xmm1 = x3 + x4 | x1 + x2 | y3 + y4 | y1 + y2		HADDPS xmm1, xmm2/m128
pshufd	Shuffle Packed Doublewords	voir exemple ci-dessous	xmmDst, xmmSrc, imm8	PSHUFD xmm1, xmm2/m128, imm8
shufps	Packed Interleave Shuffle of Quadruplets of Single-Precision Floating-Point Values			SHUFPS xmm1, xmm2/m128, imm8
unpcklps	Unpack and Interleave Low Packed Single-Precision Floating-Point Values	xmm1 = x7 | x6 | x5 | x4 | x3 | x2 | x1 xmm2 = y7 | y6 | y5 | y4 | y3 | y2 | y1 xmm1 = y5 | x5 | y4 | x4 | y3 | x3 | y2 | x2 | y1 | x1	xmmDst, xmmSrc	UNPCKLPS xmm1, xmm2/m128

AVX2

(= SSE2 avec un **V** devant les instructions + xmm en **ymm**)

Coprocesseur

8 registres (ST0 à ST7) de **80 bits** que l'on utilise comme une pile (voire une file)

le coprocesseur comporte un registre de status qui indique les erreurs éventuelles

les nombres sont chargés au niveau des registres grâce à des instructions spécifiques

les instructions liées au coprocesseur commencent par la lettre **F**

Instructions

Nom	Signification	Description	Arguments	Exemple
		Chargement		
fld	Load Floating Point Value	charge un nombre en virgule flottante depuis la mémoire et le stocke au sommet de la pile	[mem]	
fild	Load Integer	de même, avec un nombre entier qui est convertit en virgule flottante		
fldz	Load Constant	charge la valeur 0 ds **ST0**		
fld1		charge la valeur 1 ds **ST0**		
fldpi		charge la valeur de π (Pi) ds **ST0**		
		Stockage		
fst	Store Floating Point Value	stocke le sommet de pile ST0 en mémoire ou dans un autre registre du coprocesseur	[mem]	
fstp		agit comme FST mais la valeur en sommet de pile est dépilée		
fist	Store Integer	comme FST mais convertit le nombre en virgule flottante en un entier		
fistp		comme FSTP pour les entiers		
		Manipulation de la pile		
fxch	Exchange Register Contents	échange les valeurs de ST0 et STn	STn	
ffree	Free Floating-Point Register	libère un registre de la pile en le marquant comme inutilisé		
		Addition		
fadd	Add	ST0 += src, ou src est un nombre en mémoire ou un autre registre STn STn += ST0	[mem]/STi STn, ST0	
faddp		dst += ST0, puis ST0 est dépilé STn += ST0, puis ST0 est dépilé	STn STn, ST0	faddp stn, st0
fiadd		ST0 += (float) src, ou src est un entier	[mem]/STi	
		Soustraction, multiplication, division (**meme schema que l'addition**)		
fsub	Subtract	**meme schema que l'addition** **f... / f...p / fi...**		
fmul	Multiply			
fdiv	Divide			
		Comparaison		
fcom	Compare Floating Point Values	Compare **ST0** avec **src**	src	
fcomp		Compare **ST0** avec **src**, puis ST0 est dépilé		
fcompp		Compare **ST0** avec **ST1** -> ST0 & ST1 sont dépilés		
ficom	Compare Integer	Compare **ST0** avec **src**	src	
ficomp		Compare **ST0** avec **src**, puis ST0 est dépilé		
ftst	Test Floating Point Value	Compare **ST0** avec **0.0**		
fcomi	Compare Floating Point Values and Set EFLAGS	Compare **ST0** avec **STn** & set le flag en conséquence	STn	
fcomip		Compare **ST0** avec **src** & set le flag en conséquence, puis ST0 est dépilé	src	
		Autres fonctions		
fchs	Change Sign	changement de signe **ST0 = -ST0**		
fcos/sin/tan	cos / sin / tan	remplace **ST0** par son sinus / cosinus / tangente		
fabs	Absolute Value	valeurs absolue de **ST0**		
fsqrt	Square Root	racine carrée de **ST0**		