ISEL

INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Intel 386

Resumo das instruções

CENTRO DE CÁLCULO

DEPARTAMENTO DE ENGENHARIA DE ELECTRÓNICA E TELECOMUNICAÇÕES E DE COMPUTADORES

GRUPO DE SISTEMAS DEDICADOS

SECÇÃO DE MICROPROCESSADORES

Data transfer

sintaxe		descrição	Flags ODITSZAPC	exemplo
MOV dst,src reg,reg reg,mem mem,reg reg,imd mem,imd seg-reg,reg16 seg-reg,mem16 reg16,seg-reg mem,seg-reg	8 16p 32	dst = src	---------	 MOV EAX,ECX MOV EBP,stack_top MOV count[DI],CX MOV CL,2 MOV msk[BX],2CH MOV ES,CX MOV DS,seg_base MOV BP,SS MOV [EBX]save,CS

MOVSX dst,src
reg16, reg8
reg32, reg8
reg32, reg16
reg16, mem8
reg32, mem8
reg32, mem16
	8
	16p
	32
		low(dst) = low(src)
		if (high_bit(src) == 0) high(dst) = 0
		else high(dst) = ~0
			---------- 	
				MOVSX CX, DL
				MOVSX EAX, AL
				MOVSX EDX, DX
				MOVSX AX, char
				MOVSX EAX, byte ptr[ESI]
				MOVSX ECX, word ptr[EDI]

MOVZX dst,src reg16, reg8 reg32, reg8 reg32, reg16 reg16, mem8 reg32, mem8 reg32, mem16	8 16p 32	low(dst) = low(src) high(dst) = 0	----------	 MOVZX CX, DL MOVZX EAX, AL MOVZX EDX, DX MOVZX AX, char MOVZX EAX, byte ptr[ESI] MOVZX ECX, word ptr[EDI]
PUSH src Reg Seg-reg Mem	16p 32	ESP = ESP + 2 or ESP = ESP + 4 SS:[ESP] = src (PUSH CS legal)	---------	 PUSH SI PUSH ES PUSH retcode[SI]

POP dst
reg16
seg-reg
mem16
	16p
	32
		Dst = SS:[ESP];
		ESP = ESP + 2 or ESP = ESP + 4
		
		(POP CS ilegal)
			--------- 	
				POP DX
				POP DS
				POP parameter

PUSHA	16	temp = SP PUSH AX,CX,DX,BX,temp,BP,SI,DI	---------	 PUSHA
POPA	16	POP DI,SI,BP ADD ESP,2 POP BX,DX,CX,AX	---------	 POPA
PUSHAD	32	temp = ESP PUSH EAX,ECX,EDX,EBX,temp,EBP,ESI,EDI	---------	 PUSHA
POPAD	32	POP EDI,ESI,EBP ADD ESP,2 POP EBX,EDX,ECX,EAX	---------	 POPA

XCHG dst, src
mem, reg
reg, reg
	8
	16p
	32
		temp = dst
		dst = src
		src = temp
			--------- 	
				XCHG semaphore,AX
				XCHG AL,BL

IN acc, port acc, imd8 acc, DX	8 16p 32	input byte, word or dword acc = [port]	---------	 IN AL,0FAH IN AX,DX
OUT port, acc imd8, acc DX, acc	8 16p 32	output byte, word or dword [port] = acc	---------	 OUT 44,AX OUT DX,AL
LEA dst, src reg, mem	16p 32	Load Effective Address dst = address(src)	---------	 LEA BX,[BP][DI]

LDS dst,src
reg16,mem32
reg32,mem48
	16p
	32
		Load pointer
		DS = [src + 2]; dst = [src]
		DS = [src + 4]; dst = [src]
			--------- 	
				LDS SI,32_point
				LDS ESI, 48_point

LES dst,src
reg16,mem32
reg32,mem48
	16p
	32
		Load pointer
		ES = [src + 2]; dst = [src]
		ES = [src + 4]; dst = [src]
			--------- 	
				LES DI,[BX]buff
				LES EDI,[EBX]buff

LFS dst,src
reg16,mem32
reg32,mem48
	16p
	32
		Load pointer
		FS = [src + 2]; dst = [src]
		FS = [src + 4]; dst = [src]
			--------- 	
				LFS BX,[BX]ptr
				LFS EBX,[EBX]ptr

LGS dst,src
reg16,mem32
reg32,mem48
	16p
	32
		Load pointer
		GS = [src + 2]; dst = [src]
		GS = [src + 4]; dst = [src]
			--------- 	
				LGS BX,[BX]ptr
				LGS EBX,[EBX]ptr

LSS dst,src
reg16,mem32
reg32,mem48
	16p
	32
		Load pointer
		SS = [src + 2]; dst = [src]
		SS = [src + 4]; dst = [src]
			--------- 	
				LSS SP,stack_save
				LSS ESP,stack_save

Flag Control

sintaxe		descrição	Flags ODITSZAPC	exemplo
LAHF	8	AH = EFLAGS & 0xFF 7 6 4 2 0 = S Z A P C	---------	LAHF

SAHF	8	EFLAGS |= AH & 0xD5 S Z A P C = 7 6 4 2 0	----RRRRR	SAHF
PUSHF	16	SP = SP - 2 SS:[ESP] = EFLAGS	---------	PUSHF

POPF	16	EFLAGS = SS:[ESP]
		ESP = ESP + 2
			RRRRRRRRR	POPF

PUSHFD	32	ESP = ESP - 4 SS:[ESP] = EFLAGS	---------	PUSHFD

POPFD	32	EFLAGS = SS:[ESP]
		ESP = ESP + 4
			RRRRRRRRR	POPFD

CLC		CF = 0	--------0	CLC
CMC		CF = ~CF	--------X	CMC
STC		CF = 1	--------1	STC
CLD		DF = 0	-0-------	CLD
STD		DF = 1	-1-------	STD
CLI		IF = 0	--0------	CLI
STI		IF = 1 depois de executar a próxima instrução	--1------	STI
CLTS		BIT(CR0, 3) = 0	---------	CLTS

Arithmetic

sintaxe		descrição	Flags ODITSZAPC	exemplo
ADD dst,src reg,reg reg,mem mem,reg reg,imd mem,imd	8 16p 32	Dst = dst + src	X---XXXXX	 ADD DX,CX ADD DI,[BX].alfe ADD temp,CL ADD CL,2 ADD alpha,2
ADC dst,src reg,reg reg,mem mem,reg reg,imd mem,imd	8 16p 32	dst = dst + src + CF	X---XXXXX	 ADC AX,SI ADC DX,beta[SI] ADC key[SI],DI ADC BX,256 ADC gamma,30H
INC dst reg mem	8 16p 32	dst=dst+1	X---XXXXX	 INC BL INC alpha[DI]
AAA		Ajuste para ASCII após adição	U---UUXUX	AAA
DAA		Ajuste para BCD após adição, no AL	X---XXXXX	DAA
SUB dst,src reg,reg reg,mem mem,reg reg,imd mem,imd	8 16p 32	dst = dst - src	X---XXXXX	 SUB BX,CX SUB DX,math[SI] SUB [BP+2],CL SUB SI,5280 SUB [BP]yes,1000
SBB dst,src reg,reg reg,mem mem,reg reg,imd mem,imd	ec	Subtracção com o borrow dst = dst - src - CF	X---XXXXX	 SBB BX,CX SBB DI,[BX]pay SBB balance,AX SBB CL,1 SBB count[SI],10
DEC dst Reg mem	8 16p 32	dst = dst - 1	X---XXXXX	 DEC AL DEC array[SI]
NEG dst reg mem		dst = -dst	XXXXXXXXX	 NEG AL NEG multiplier
CMP dst,src reg,reg reg,mem mem,reg reg,imd mem,imd	8 16p 32	Flags modificadas de acordo com o resultado da operação dst - src	X---XXXXX	 CMP BX,CX CMP DH,alpha CMP [BP+2],SI CMP BL,02H CMP [BX]x,3420H
AAS		Ajuste ASCII após subtracção em AL	U---UUXUX	AAS
DAS		Ajuste BCD após subtracção em AL	X---XXXXX	DAS
MUL src reg8 mem8 reg16 mem16 reg32 mem32	8 16p 32	Multiplicação de números sem sinal AX = AL * byte DX:AX = AX * word EDX:EAX = EAX * dword	X---UUUUX	 MUL BL MUL month[SI] MUL CX MUL baund_rate MUL BX MUL dword ptr[ESI]

IMUL
op1,[op2,[op3]]
  reg8
  mem8
 	
  reg16
  mem16
 	
  reg32
  mem32
 	
  reg, reg
  reg, mem
  reg, imd
 	
  reg, reg, imd
  reg, mem, imd
	8
	16p
	32
		Multiplicação de números com sinal
		
		AL = AL * byte
		
		DX:AX = AX * word
		
		EDX:EAX = EAX * dword
		
		op1 = op1 * op2
		
		op1 = op2 * op3
			X---UUUUX	
				IMUL CL
				IMUL rate_byte
				
				IMUL BX
				IMUL red[BP][DI]
				
				IMUL EBX
				IMUL dword ptr[ESI]
				
				IMUL ECX, 23
				
				IMUL AL, CH, 7

AAM		Ajuste para ASCII após multiplicação	U---XXUXU	AAM

DIV src
  reg8
  mem8
 	
  reg16
  mem16
 	
  reg32
  mem32
	8
	16p
	32
		Divisão de números sem sinal
		AL = AX / byte;
		AH = AX % byte
		
		AX = DX:AX / word;
		DX = DX:AX % word
		
		EAX = EDX:EAX / dword;
		EDX = EDX:EAX % dword
			U---UUUUU	
				DIV CL
				DIV alpha
				
				DIV BX
				DIV table[SI]
				
				DIV EBX
				DIV dword ptr[ESI]

IDIV src
  reg8
  mem8
 	
  reg16
  mem16
 	
  reg32
  mem32
	8
	16p
	32
		Divisão de números com sinal
		AL = AX / byte;
		AH = AX % byte
		
		AX = DX:AX / word;
		DX = DX:AX % word
		
		EAX = EDX:EAX / dword;
		EDX = EDX:EAX % dword
			U---UUUUU	
				IDIV BL
				IDIV div_byt[SI]
				
				IDIV CX
				IDIV [BX]dado
				
				IDIV EBX
				IDIV dword ptr[ESI]

AAD		Ajuste para ASCII antes da divisão	U---XXUXU	AAD

CBW	8	Estende o sinal de AL para AX
		if (AL < 80h) AH = 0FFh;
		else AH = 0;
			---------	CBW

CWD	16	Estende o sinal de AX para DX
		if (AX < 8000h) DX = 0FFFFh;
		else DX = 0;
			---------	CWD

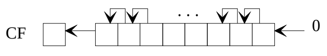
CDQ	32	Estende o sinal de EAX para EDX
		if (EAX < 8000h) EDX = 0FFFFFFFFh;
		else EDX = 0;
			---------	CDQ

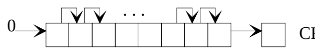
Logic

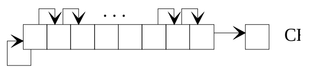
sintaxe		descrição	Flags ODITSZAPC	exemplo

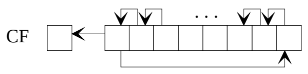
SHLD dst,src,count
  reg, reg, imd
  mem, reg, imd
  reg, reg, CL
  mem, reg, CL
	16p
	32
		temp = count & 1fh
		value = concatenate(dest, src)
		value = value << temp
		dst = value
			---------	MOV EAX, [ESI]
				SHLD [ESI+4], EAX, 7

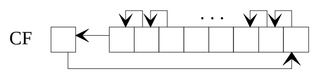
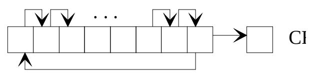
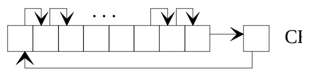
SHRD dst,src,count
  reg, reg, imd
  mem, reg, imd
  reg, reg, CL
  mem, reg, CL
	16p
	32
		temp = count & 1fh
		value = concatenate(src, dest)
		value = value >> temp
		dst = value
			---------	MOV EAX, [ESI]
				SHLD [ESI+4], EAX, 7

SAL/SHL dst,count
  reg,CL
  reg,imd5
  mem,CL
  mem,imd5
	8
	16p
	32
			X-------X	
				SAL DI,CL
				SAL AX,5
				SAL stor_cnt,CL
				SAL [BX]status,3

SHR dst,count
  reg,CL
  reg,imd5
  mem,CL
  mem,imd5
	8
	16p
	32
			X-------X	
				SHR SI,CL
				SHR SI,1
				SHR input,CL
				SHR by[SI][BX],1

SAR dst,count
  reg,CL
  reg,imd5
  mem,CL
  mem,imd5
	8
	16p
	32
			X-------X	
				SAR DI,CL
				SAR DX,1
				SAR n_blocks,CL
				SAR n_blocks,1

ROL dst,count
  reg,CL
  reg,imd5
  mem,CL
  mem,imd5
	8
	16p
	32
			X-------X	
				ROL DI,CL
				ROL BX,1
				ROL alpha,CL
				ROL byte[DI],2

RCL dst,count reg,CL reg,imd5 mem,CL mem,imd5	8 16p 32		X-------X	 RCL AL,CL RCL CX,1 RCL [BP]parm,CL RCL alpha,4
ROR dst,count reg,CL reg,imd5 mem,CL mem,imd5	8 16p 32		X-------X	 ROR BX,CL ROR AL,1 ROR cmd_word,CL ROR port_stat,1
RCR dst,count reg,CL reg,imd5 mem,CL mem,imd5	8 16p 32		X-------X	 RCR BL,CL RCR BX,10 RCR array[DI],CL RCR dword ptr[ESI], 24
AND dst,src reg,reg reg,mem mem,reg reg,imd mem,imd	8 16p 32	dst = dst & src (and bit a bit)	0---XXUX0	 AND AL,BL AND CX,flag_word AND ascii[DI],AL AND CX,0F0H AND beta,03H
TEST dst,src reg,reg reg,mem reg,imd mem,imd	8 16p 32	Flags modificadas de acordo com a operação dst & src (bit a bit)	0---XXUX0	 TEST SI,DI TEST SI,end_cnt TEST BX,0CC4H TEST retcode,01H
OR dstsrc reg,reg reg,mem mem,reg reg,imd mem,imd	8 16p 32	dst = dst | src (or bit a bit)	0---XXUX0	 OR AL,BL OR DX,prtid[DI] OR flag_byte,CL OR CX,01H OR [BX]car,0CFH
XOR dstsrc reg,reg reg,mem mem,reg reg,imd mem,imd	8 16p 32	dst = dst ^ src (xor bit a bit)	0---XXUX0	 XOR CX,BX XOR CL,mask_byte XOR alpha[SI],DX XOR SI,00C2H XOR retcode,0D2H
NOT dst reg mem	8 16p 32	dst= ~dst (inverte bit a bit)	---------	 NOT AX NOT charater

String manipulation

sintaxe		descrição	Flags ODITSZAPC	exemplo
REP		CX = CX - 1; Repete operação de string enquanto CX <> 0	---------	REP MOVS dst,src
REPE/REPZ		CX = CX - 1; Repete operação de string enquanto CX <> 0 && ZF == 1	---------	REPE CMPS ok,key
REPNE/REPNZ		CX = CX - 1; Repete operação de string enquanto CX <> 0 && ZF == 0	---------	REPNE CMPS up,ok
MOVS dst_s,src_s MOVSB MOVSW MOVSD REP MOVS	8 16p 32	Move strings Byte n = 1; word n = 2; dword n = 4 ES:[EDI] = DS:[ESI] If (DF == 0) {ESI += n; EDI += n} Else {ESI -= n; EDI -= n}	---------	MOVS blk1, blk2 REP MOVSB
CMPS dst,src CMPSB CMPSW CMPSD REP CMPS	8 16p 32	Compara strings Byte n = 1; word n = 2; dword n = 4 ES:[EDI] - DS:[ESI] If (DF == 0) {ESI += n; EDI += n} Else {ESI -= n; EDI -= n}	X---XXXXX	CMPS blk1, blk2 REP CMPSB
SCAS dst-string SCASB SCASW SCASD REP SCAS	8 16p 32	Scan string Byte n = 1; word n = 2; dword n = 4 acc - ES:[EDI] if (DF == 0) EDI += n; else EDI -= n	X---XXXXX	SCAS REPNE SCAS

LODS src-string
LODSB
LODSW
LODSD
REP LODS
	8
	16p
	32
		Load string
		byte n = 1; word n = 2; dword n = 4
		acc = DS:[ESI]
		if (DF == 0) ESI += n; else ESI -= n
			---------	LODS tab
				
				REP LODS

STOS dst-string	8	Store string	---------	STOS tab
STOSB	16p	byte n = 1; word n = 2; dword n = 4		
STOSW	32	ES:[EDI] = acc		
STOSD		if (DF == 0) EDI += n; else EDI -= n		REP STOS
REP STOS				
INS dst-string	8	Input string from I/O port	---------	INS tab
INSB	16p	byte n = 1; word n = 2; dword n = 4		
INSW	32	ES:[EDI] = port(DX)		
INSD		if (DF == 0) EDI += n; else EDI -= n		REP INSB
REP INS				
OUTS dst-string	8	Output string to I/O port	---------	OUTS tab
OUTSB	16p	byte n = 1; word n = 2; dword n = 4		
OUTSW	32	port(DX) = ES:[EDI]		
OUTSD		if (DF == 0) EDI += n; else EDI -= n		REP OUTSB
REP OUTS				
XLAT scr-table	8	AL = ES:[EBX+AL]	---------	XLAT ascii_tab

Bit manipulation

sintaxe		descrição	Flags ODITSZAPC	exemplo
BSF dst, src	8	Scan bit forward	U---UXUUU	
reg, reg	16p	for(i = 0; bit(src, i) == 0; i++);		
reg, mem	32	dst = i;		
BSR dst, src	8	Scan bit reverse	U---UXUUU	
reg, reg	16p	for(i = 15(31); bit(src,i)==0; i--);		
reg, mem	32	dst = i;		
BT dst, index	8	Test bit	U---UUUUX	
reg, imd	16p	CF = bit(dst, index)		
mem, imd	32			
reg, reg				
mem, reg				
BTC dst, index	8	Test bit and complement	U---UUUUX	
reg, imd	16p	CF = bit(dst, index)		
mem, imd	32	bit(dst, index) = ~ bit(dst, index)		
reg, reg				
mem, reg				
BTR dst, index	8	Test bit and reset	U---UUUUX	
reg, imd	16p	CF = bit(dst, index)		
mem, imd	32	bit(dst, index) = 0		
reg, reg				
mem, reg				
BTS dst, index	8	Test bit and set	U---UUUUX	
reg, imd	16p	CF = bit(dst, index)		
mem, imd	32	bit(dst, index) = 1		
reg, reg				
mem, reg				

Control transfer

sintaxe		descrição	Flags ODITSZAPC	exemplo
CALL target			---------	
near-proc		PUSH(EIP); EIP = near-proc		CALL NEAR_PROC
far-proc		PUSH(CS); PUSH(EIP); CS:EIP=far-proc		CALL FAR_PROC
reg32		PUSH(EIP); EIP = reg32		CALL AX
mem32		PUSH(EIP); EIP = [mem]		CALL WORD PTR ...
mem48		PUSH(CS); PUSH(EIP); CS:EIP =		CALL DWORD PTR ..
		selector:dword ptr[mem]		
JMP target		Jump incondicional	---------	
short-label		EIP += deslocamento (8 bits)		JMP short
near-label		EIP = deslocamento (32 bits)		JMP within_seg
far-label		CS:EIP = far-label		JMP far_label
reg32		EIP = reg32		JMP CX
mem32		EIP = [mem]		JMP [BX]target
mem48		CS:EIP = selector:dword ptr[mem]		JMP new.seg[SI]
RET count (16 bit)			---------	
near,no pop		POP(EIP)		RET
near,pop		POP(EIP); ESP = ESP + count		RET 4
far,no pop		POP(EIP); POP(CS)		RET
far,pop		POP(EIP); POP(CS); ESP = ESP + count		RET 2

ENTER
  locals, nesting
  imd16, imd8
		nesting = nesting & 1fh
		push(EBP)
		temp = ESP
		if (nesting > 0) {
		    nesting--;
		    for (nesting --;
		         nesting > 0; nesting--) {
		        EBP = EBP – 4
		        push(SS:[EBP])
		    }
		    push (temp)
		}
		EBP = temp
		ESP = ESP – locals
			--------- 		

LEAVE		MOV ESP, EBP POP EBP	---------	

JG/JNLE disp
  disp8
  disp32
	8	Jump if greater / not less nor equal
		if (CF == OF && ZF == 0) EIP += disp
		(operandos com sinal)
			---------	JG greater

JGE/JNL disp
  disp8
  disp32
	8	Jump if greater or equal / not less
		if (CF == OF) EIP += disp
		(operandos com sinal)
			---------	JGE great_equal

JL/JNGE disp
  disp8
  disp32
	8	Jump if less / not greater nor equal
		if (CF != OF) EIP += disp
		(operandos com sinal)
			---------	JL less

JLE/JNG disp
  disp8
  disp32
	8	Jump if less or equal / not greater
		if (CF != OF || ZF == 1) EIP += disp
		(operandos com sinal)
			---------	JLE not_above

JA/JNBE disp
  disp8
  disp32
	8	Jump if above / not below nor equal
		if (CF == 0 && ZF == 0) EIP += disp
		(operandos sem sinal)
			---------	JA above

JAE/JNB disp
  disp8
  disp32
	8	Jump if above or equal / not below
		if (CF == 0) EIP += disp
		(operandos sem sinal)
			---------	JAE above_equal

JB/JNAE disp
  disp8
  disp32
	8	Jump if below / not above nor equal
		if (CF == 1) EIP += disp
		(operandos sem sinal)
			---------	JB below

JBE/JNA disp
  disp8
  disp32
	8	Jump if below or equal / not above
		if (CF == 1 || ZF == 1) EIP += disp
		(operandos sem sinal)
			---------	JNA not_above

JP/JPE disp
  disp8
  disp32
	8	Jump if parity / parity even
		if (PF == 1) EIP += disp
			---------	JPE even_parity

JNP/JPO disp
  disp8
  disp32
	8	Jump if not parity / parity odd
		if (PF == 0) EIP += disp
			---------	JPO odd_parity

JO disp
  disp8
  disp32
	8	Jump if overflow
		if (OF == 1) EIP += disp
			---------	JO overflow

JNO disp
  disp8
  disp32
	8	Jump if not overflow
		if (OF == 0) EIP += disp
			---------	JNO no_overflow

JS disp
  disp8
  disp32
	8	Jump if sign
		if (SF == 1) EIP += disp
			---------	JS negative

JNS disp
  disp8
  disp32
	8	Jump if not sign
		if (SF == 0) EIP += disp
			---------	JNS positive

JE/JZ disp
  disp8
  disp32
	8	Jump if equal / zero
		if (ZF == 1) EIP += disp
			---------	JZ zero

JNE/JNZ disp
  disp8
  disp32
	8	Jump if not equal / not zero
		if (ZF == 0) EIP += disp
			---------	JNE not_equal

JC disp
  disp8
  disp32
	8	Jump if carry
		if (CF == 1) EIP += disp
			---------	JC carry_set

JNC disp
  disp8
  disp32
	8	Jump if not carry
		if (CF == 0) EIP += disp
			---------	JNC not_carry

JCXZ disp disp8	8	Jump if CX is zero if (CX == 0) EIP += disp	---------	JCXZ count_done
JECXZ disp disp8	8	Jump if ECX is zero if (ECX == 0) EIP += disp	---------	JCXZ count_done

LOOP disp8	8	CX = CX - 1;
		if (CX != 0) EIP += disp
			---------	LOOP again

LOOPE/LOOPZ disp
  disp8
	8	CX = CX - 1;
		if (CX != 0 && ZF == 1) EIP += disp
			---------	LOOPE again

LOOPNE/LOOPNZ disp disp8	8	CX = CX - 1; If (CX != 0 && ZF == 0) EIP += disp	---------	LOOPNE again

Interrupt instructions

sintaxe		descrição	Flags ODITSZAPC	exemplo

INT interrupt-type
    type == 3
    type != 3
	8	PUSH(EFLAGS);PUSH(CS);PUSH(EIP);
		TF = 0;
		if (IDT[vector].TYPE == INT_GATE)IF=0
		CS:EIP = destination(IDT[vector])
			--00-----	
				INT 3
				INT 67

INTO		if (OF == 1)
		    INT 4
			--00-----	INTO

BOUND dst, src
    reg16, mem32
    reg32, mem64
	16p
	32
		if (reg16 < word ptr [mem] ||
		    reg16 > word ptr [mem + 2])
		    INT 5
		if (reg32 < dword ptr [mem] ||
		    reg32 > dword ptr [mem + 4])
		    INT 5
				VC_LIM:
				    DD 1, 20
				VC DD 20 DUP(?)
				
				MOV EAX, [EBP-6]
				BOUND EAX, VC_LIM

IRET		Return interrupt
		if (NT == 1)
		    TASK_RETURN(TSS, back_link)
		else
		    POP(EIP); POP(CS); POP(EFLAGS)
			RRRRRRRRR	IRET

Conditional byte set

sintaxe		descrição	Flags ODITSZAPC	exemplo

SETG/SETNLE dst
    reg8
    mem8
	8	Set byte greater / not less or equal
		dst = (CF == OF && ZF == 0)
		(operandos com sinal)
			---------	SETG AL

SETGE/SETNL dst
    reg8
    mem8
	8	Set byte greater or equal / not less
		dst = (CF == OF)
		(operandos com sinal)
			---------	SETGE AL

SETL/SETNGE dst
    reg8
    mem8
	8	Set byte less / not greater nor equal
		dst = (CF != OF)
		(operandos com sinal)
			---------	SETL AL

SETLE/SETNG dst
    reg8
    mem8
	8	Set byte less or equal / not greater
		dst = (CF != OF || ZF == 1)
		(operandos com sinal)
			---------	SETLE AL

SETA/SETNBE dst
    reg8
    mem8
	8	Set byte above / not below nor equal
		dst = (CF == 0 && ZF == 0)
		 (operandos sem sinal)
			---------	SETA AL

SETAE/SETNB dst
    reg8
    mem8
	8	Set byte above or equal / not below
		dst = (CF == 0)
		(operandos sem sinal)
			---------	SETAE AL

SETB/SETNAE dst
    reg8
    mem8
	8	Set byte below / not above nor equal
		dst = (CF == 1)
		(operandos sem sinal)
			---------	SETB AL

SETBE/SETNA dst
    reg8
    mem8
	8	Set byte below or equal / not above
		if (CF == 1 || ZF == 1)
		(operandos sem sinal)
			---------	SETBE AL

SETP/SETPE dst
    reg8
    mem8
	8	Set byte on parity / parity even
		if (PF == 1)
			---------	SETP AL

SETNP/SETPO dst
    reg8
    mem8
	8	Set byte on not parity / parity odd
		if (PF == 0)
			---------	SETNP AL

SETO dst
    reg8
    mem8
	8	Set byte on overflow
		dst = (OF == 1)
				SETO AL

SETNO dst
    reg8
    mem8
	8	Set byte on not overflow
		dst = (OF == 0)
				SETNO AL

SETS dst
    reg8
    mem8
	8	Set byte on sign
		dst = (SF == 1)
			---------	SETS AL

SETNS dst
    reg8
    mem8
	8	Set byte on not sign
		dst = (SF == 0)
			---------	SETNS AL

SETE/SETZ dst
    reg8
    mem8
	8	Set byte on equal / zero
		dst = (ZF == 1)
			---------	SETE AL

SETNE/SETNZ dst
    reg8
    mem8
	8	Set byte on not equal / not zero
		dst = (ZF == 0)
			---------	SETNE AL

SETC dst reg8 mem8	8	Set byte on carry dst = (CF == 1)	---------	SETC AL
SETNC dst reg8 mem8	8	Set byte on not carry dst = (CF == 0)	---------	SETNC AL

Processor control

sintaxe		descrição	Flags ODITSZAPC	exemplo
HLT		Halt	---------	HLT
WAIT		Espera a activação do sinal do TEST#	---------	WAIT
NOP		Nao executa operacao		NOP
MOV dst, src CR, reg32 reg32, CR DR, reg32 reg32, DR TR, reg32 reg32, TR		Move especial		 MOV CR0, EAX MOV EAX, CR1
ESC opcode,src imd,mem imd,reg		Para o co-processador Data_bus = src	---------	 ESC 6,array[SI] ESC 20,AL
LOCK		Activa o sinal LOCK durante a execução da proxima instrução	---------	LOCK XCHG flg,AL

Protection control

sintaxe		descrição	Flags ODITSZAPC	exemplo

ARPL dst. src
reg16, reg16
mem16, reg16
	16	Adjust RPL Field of Selector
		If (dst.RPL < src.RPL)
		dst.RPL = src.RPL
		ZF = 1
		Else
		ZF = 0
			-----X---	MOV EAX,[EBP+12]
				ARPL EAX,[EBP+2]
				JNZ bad_param

LAR dst, selector reg16, reg16 reg16, mem16 reg32, reg16 reg32, mem16	16p 32	Load Access Rights If (check_access(selector)) dst = descriptor(selector).access_rigths & 00F?FF00h ZF = 1 Else ZF = 0	-----X---	

LGDT op
mem48
		Load GDT register
		GDTR.limit = [op]
		GDTR.base = [op + 2]
			---------	LGDT init_table

LIDT op
mem48
		Load IDT register
		IDTR.limit = [op]
		IDTR.base = [op + 2]
			---------	IGDT int_table

LLDT op reg16 mem16	16	Load LDT register LDTR = op	---------	LLDT task_b,ldtr
LMSW op reg16 mem16	16	Load machine status word CR0 = (CR0 & FFFF0000h) | op	---------	LMSW init_state

LSL dst, selector
reg32, reg16
reg32, mem16
	16p
	32
		Load segment limit
		if (access_ok(selector))
		dst = descriptor(select).limit
		ZF = 1
		else
		ZF = 0
			-----X---	LSL EAX, [BP + 2]

LTR selector reg16 mem16	16	Load task register TR = selector	---------	LTR AX
SGDT dest mem48		Store GDT register [dst] = GDTR.limit [dst + 2] = GDTR.base	---------	SGDT [300h]
SIDT dest mem48		Store IDT register [dst] = IDTR.limit [dst + 2] = IDTR.base	---------	IGDT int_tab
SLDT dst reg16 mem16	16	Store LDT register dst = LDTR	---------	SLDT DX
SMSW op Reg16 Mem16	16	Store machine status word dst = MSW low16(CR0)	---------	SMSW [DI]

STR dst
    reg16
    mem16
	16	Store task register
		dst = TR
			---------	STR CX

VERR selector
    reg16
    mem16
	16	Verify read access
		if (accessible(selector) &&
		    read_access(selector))
		    ZF = 1
		else
		    ZF = 0
			-----X---	VERR word ptr [EBP+8]
				JZ continue
				STC
				LEAVE
				RETF

VERW selector
    reg16
    mem16
	16	Verify write access
		if (accessible(selector) &&
		    write_access(selector))
		    ZF = 1
		else
		    ZF = 0
			-----X---	VERW word ptr [EBP+8]
				JZ continue
				STC
				LEAVE
				RETF

mem	conteúdo de memória 8, 16 ou 32 bits
reg	conteúdo de registo 8, 16 ou 32 bits
acc	AL, AX ou EAX
mem16	conteúdo de memória a 16 bits
reg16	conteúdo de registo a 16 bits
seg-reg	registo de selector
mem32	conteúdo de memória a 32 bits
imd8	valor imediato a 8 bits
imd16	valor imediato a 16 bits
short-label	label a uma distância de +127 a -128 da posição corrente
near-label	label dentro do segmento corrente
far-label	label fora do segmento corrente
near-proc	procedimento dentro do segmento corrente
far-proc	procedimento fora do segmento corrente
disp8	deslocamento a 8 bit
disp32	deslocamento a 32 bit

-	não alterada
0	posta a zero
1	posta a um
x	afetada de acordo com o resultado
u	indefinida
R	retoma valor salvo

Formato do registo EFlags

														VM	RF	0	NT	IOPL		OF	DF	IF	TF	SF	ZF	0	AF	0	PF	1	CF
31														17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0