

# Høgskulen på Vestlandet

Institutt for datateknologi, elektroteknologi og realfag

Eksamen i: DAT103 – Datamaskiner og operativsystem  
Dato: 13 juni 2023  
Tid: 9:00 – 13:00  
Lovlige hjelpemiddel: vanlig kalkulator  
Språk: norsk bokmål

Dette oppgavesettet består av 6 sider, bortsett fra vedlegget.  
Du kan svare enten på norsk eller engelsk.

*Lykke til!  
Dag og Violet*

---

## Oppgave 1 – Flervalgsspørsmål

**(24%)**

*For hvert av spørsmålene nedenfor, velg ikke mer enn et av alternativene.*

- 1.1 Hvilke av de følgende tillater I/O moduler og hovedminnet å utveksle data direkte?
  - (a) Memory management unit (MMU)
  - (b) Direct memory access (DMA)
  - (c) Memory address register
  - (d) Alle de ovenfor
  - (e) Ingen av de ovenfor
- 1.2 Hvilke av de følgende kan forklare hvorfor mer hovedminne (RAM) kan øke datamaskinen sin ytelse?
  - (a) Mer RAM kan redusere den eksterne fragmenteringen
  - (b) Mer RAM kan øke prosessorhastigheten
  - (c) Mer RAM kan redusere sannsynligheten for sidefeil
  - (d) Alle de ovenfor
  - (e) Ingen av de ovenfor
- 1.3 Hvilken av de følgende vil bli lagret under et *kontekstbytte* mellom prosesser
  - (a) Stack pointer
  - (b) Program counter
  - (c) CPU registers
  - (d) Alle de ovenfor
  - (e) Ingen av de ovenfor

- 1.4 Hvilke av de følgende datastrukturene er generelt brukt av operativsystem for å lagre informasjon knyttet til en prosess?
- (a) Process control block (PCB)
  - (b) Process identifier
  - (c) Process state
  - (d) Alle de ovenfor
  - (e) Ingen av de ovenfor
- 1.5 En adresse generert av CPUen er vanlegvis omtalt som en ...
- (a) Physical address
  - (b) Absolute address
  - (c) Logical address
  - (d) Alle de ovenfor
  - (e) Ingen av de ovenfor
- 1.6 Hvilke av de følgende er en fordel med virtuelt minne?
- (a) Det tillater utførelse av prosesser som ikke er fullstendig i minnet
  - (b) Det abstraherer hovedminnet til et ekstremt stort logisk minne
  - (c) Det tillater å kjøre program som er større enn fysisk minne
  - (d) Alle de ovenfor
  - (e) Ingen av de ovenfor
- 1.7 Hvilken trådmodell tillater alle tråder å kjøre på same tid, og samtidig alle CPU-kjerner å bli utnyttet på same tid?
- (a) En-til-en
  - (b) En-til-mange
  - (c) Mange-til-en
  - (d) Mange-til-mange
  - (e) Ingen av de ovenfor
- 1.8 Hva er en kjernetråd?
- (a) En virtuell CPU som bruker-tråder er tilordnet
  - (b) Det som tråder blir kalt når brukt av kernelen
  - (c) Gamelt navn for tråder
  - (d) Bare bruke for en-til-en tråd-modell
- 1.9 Hvorfor spiller det noen rolle at mobiltelefonen din er en Von Neumann-maskin?
- (a) Det spiller ingen rolle i det hele tatt
  - (b) Det gjør det mulig å installere program fra app-butikker
  - (c) Det betyr at den kan utføre alle operasjoner (Von Neumann-komplett)
  - (d) Mobiltelefoner er generelt sett ikke Von Neumann-maskiner

- 1.10 Hva er eksternt minne?
- (a) Lagring som er utenfor datamaskin-kabinettet
  - (b) Lagring som er utenfor CPU
  - (c) Lagring som som ikke kan adresseres direkte fra CPU
  - (d) "Hot-pluggable" lagring (kan bli koblet til en kjørendee datamaskin)
- 1.11 Hva av de følgende er er riktig om en databuss?
- (a) Den har mulighet til å forbinde mer enn to enheter i et datasystem
  - (b) Det kan ikke vere seriell (må ha flere parallelle datalinjer)
  - (c) Den må ha stor båndbredde
  - (d) Den er bare brukt til å kople CPU til hovedminne
  - (e) Alle de ovenfor
- 1.12 Hva av det følgende er riktig om CPU?
- (a) En CPU-brikke kan bare ha en kjerne
  - (b) En datamaskin kan bere ha en CPU-brikke
  - (c) Om en datamaskin har mer enn en kjerne, så kan det bare være en CPU-brikke i datamaskinen
  - (d) En datamaskin kan ha flere CPU-brikker med flere kjerner hver

## Oppgave 2 – Diverse

(16%)

- 2.1 Ta utgangspunkt i et datasystem som bruker 16-bit logisk adresse. Hva er størrelsen på det logiske adresserommet?
- 2.2 Ta utgangspunkt i et datasystem som støtter "paging" maskinvare med en "translation look-aside buffer" (TLB). Anta en prosess som har *noen* av sine forespurte sider i fysisk minne. Sidetabellen for prosessen trenger å bli oppdatert når det det er en sidefeil. Anta videre at prosessen forsøker å få tilgang til en side.
- (a) Hvor mange minnetilganger er nødvendig for å hente siden om sidenummeret er funnet i TLB og det ikke er en sidefeil?
  - (b) Hvor mange minnetilganger er nødvendig for å hente siden om sidenummeret ikke er funne i TLB og det det ikke er en sidefeil?
  - (c) Hvor mange tilganger til "backing store" er nødvendig om det er en sidefeil og offersiden er skitten?
  - (d) Når en prosess bruker mer tid på "paging" enn utføring, så sier vi at prosessen ...
- 2.3 Forklar forskjellene og likhetene mellom prosesser og tråder.
- 2.4 Du har nettopp startet en ny prosess ved å bruke fork systemkallet. Hvordan kan du vite om du nå kjører i forelder eller barne-prosessen?
- 2.5 Hva betyr det at en tråd-modell er en-til-en? Hvilke andre modeller er mulige? Forklar.

## Oppgave 3 – Cache

(15%)

- 3.1 Ta utgangspunkt i at  $B_0, B_2, B_4, B_6$  and  $B_8$  er tilordnet til  $C_0$ , og  $B_1, B_3, B_5, B_7$  og at  $B_9$  er tilordnet til  $C_1$  hvor  $B_i$  refererer til  $i$ . blokk i hovedminnet og  $C_j$  refererer til  $j$ . cache-linje. Anta at  $B_1$  og  $B_4$  er i cache. Hva er antall “cache misses” om  $B_8, B_4, B_1, B_8, B_1$  har tilgang i den viste rekkefølgen? Rettferdigjør svaret ditt ved å vise hvilke tilganger som førte til “cache miss”.
- 3.2 Forklar kva som er mening med cache minne. Hva ville virkningene vært om en CPU ikke har cache minne i det hele tatt?
- 3.3 Du vil kjøpe en ny datamaskin. Du vurderer en modell med flere muligheter for mengde hovedminne og cache minne (men ellers samme spesifikasjoner). Hva er virkningene av å øke størrelsen av cache minne sammenlignet med å øke størrelsen på hovedminne?
- 3.4 Du utvikler noe programvare. En kollega ser på koden din, og seier at det vil bli mange sidefeil slik du har skrevet koden. Hva kan årsakene til dette være?
- 3.5 Typisk har en moderne CPU flere lag med cache minne. Forklar grunnen til dette - hvorfor ikke bare et enkelt lag?

## Oppgave 4 – Assembler og adressemodi

(11%)

- 4.1 Ta utgangspunkt i fragmentet med assembler-kode i Listing 1.

```
1 push dword 1
2 push dword 2
3 push dword 3
4 push dword 4
5 pop  eax
6 mov  ebx, [esp+4]
7 pop  ecx
8 add  eax, ebx
```

Listing 1

Anta at alle instruksjonene i Listing 1 er utført, og at “stack” vokser mot lavere adresser.

- (a) Hva er verdien lagret i register `ecx`?
  - (b) Hva er verdien lagret i register `ebx`?
  - (c) Hva er verdien lagret i register `eax`?
  - (d) Hva er verdien ved adressen lagret i `esp`?
  - (e) Hva er verdien ved adressen lagret i `esp+4`?
- 4.2 Identifiser hvilken adressemodus som blir brukt for hver av operandene i de følgende instruksjonene:
- (a) `sub eax, 5`
  - (b) `mov ebx, [eax]`
  - (c) `mov [esp + 4], NUM`

**Oppgave 5 – Prosess synkronisering****(12%)**

- 5.1 Typisk kan bare en prosess skrive til en *kritisk seksjon* på samme tid, mens det er mulig for flere prosesser å lese fra en kritisk seksjon på samme tid. Men hva er situasjonen om en prosess ønsker å lese mens en annen ønsker å skrive til den samme kritiske seksjonen på samme tid? Forklar.
- 5.2 Tre populære metoder for å beskytte en kritisk seksjon er *låsar*, *semaforar* og *monitorar*. Forklar forskjeller og likheter mellom disse metodene.
- 5.3 En kollega utvikler noe programvare med to prosesser som utveksler data. Denne personen har ikke brukt noen mekanisme for prosesssynkronisering, men seir at han/hun ikke har hatt noen problem med programvaren på tross av dette. Forklar for denne personen hvorfor dette ikke er noen god ide, og mulige grunner for at han/hun ikke har merket noe problem.
- 5.4 Noe programvare trenger å beskytte en delt variabel. Bare en prosess kan skrive til variabelen til samme tid, mens flere prosesser kan lese fra den til samme tid (selvsagt bare om en annen prosess ikke skriver til den). Skriv pseudo-kode for dette. Du kan anta at grunnleggende funksjoner det er behov for er tilgjengeleg fra systembibliotek.

**Oppgave 6 – CPU tidsplanlegging****(10%)**

- 6.1 Ta utgangspunkt i følgende tabell for prosesseringtid og periode for to *periodiske* prosesser  $P_1$  og  $P_2$ :

Prosess	Prosesseringtid	Periode
$P_1$	2	9
$P_2$	4	5

Tidsfristen for hver av prosessene er starten på sin neste periode. For eksempel, fristen for fullføring for  $P_1$  er ved tid 9, 18, 27, ..., mens frist for fullføring av  $P_2$  er ved tid 5, 10, 15, ...

Bruk *Første frist først* for å tidsplanlegge de to prosessene. Ved  $tid = 20$ , kan de to prosessene møte alle deres frister? Tegn Gantt-diagram av utførelsen for å rettferdiggjøre svaret.

- 6.2 Ta utgangspunkt i følgende tabell for prosesseringtid og periode for to *periodiske* prosesser  $P_1$  og  $P_2$ :

Prosess	Prosesseringtid	Periode
$P_1$	5	12
$P_2$	3	6

Fristen for hver av prosessene er starten på sin neste periode. For eksempel, fristen for fullføring for  $P_1$  er ved tid 12, 24, 36, ..., mens fristen for fullføring av  $P_2$  er ved tid 6, 12, 24, ... Bruk "*Rate-Monotonic Scheduling*" for å tidsplanlegge de to prosessene, hvor prioriteten er tilordnet basert på perioden for hver prosess: *dess kortere periode, dess høyere prioritet*. Ved  $tid = 24$ , kan de to prosessene møte alle deres frister? Tegn Gantt-diagram av utførelsen for å rettferdiggjøre svaret.

**Oppgave 7 – Sideinndeling****(12%)**

7.1 Anta at hver sidestørrelse er 2 KB<sup>1</sup>. Ta utgangspunkt i følgende tabell:

Sidenummer (i)	Rammenummer (j)	Ramme sin startadresse (k)
5	4	8192
6	12	24576
7	6	12288
8	5	10249

Hver av linjene representerer en tilordning fra side  $i$  til ramme  $j$ , hvor ramme  $j$  starter fra adresse  $k$  i det fysiske minnet. For eksempel, den første linjen refererer til side 5 er tilordnet til ramme 4 som starter fra adresse 8192 i det fysiske minnet.

- (a) Skriv ned i desimaltall, *sidenummeret*, "*offset*" og den *fysiske adressen* for den logiske adressen 14378.
- (b) Skriv ned i desimaltall, den *logiske adressen* for den fysiske adressen 9216.

7.2 Nå, anta at sidestørrelsen er 4 KB.

- (a) Om en prosess forespør 20598 byter fra operativsystemet, hvor mange sider vil prosessen bli allokert? Er det noen intern fragmentering?
- (b) Ta utgangspunkt i et datasystem som bruker 16-bit logiske adresser. Hvor mange linjer er det i en sidetabell? Hva er størrelsen for sidetabellen?

---

<sup>1</sup>1 KB er 1024 bytes

## Vedlegg – ASCII-tabell (fra [www.asciitable.com](http://www.asciitable.com))







Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	<b>NUL</b> (null)	32	20	040	&#32;	<b>Space</b>	64	40	100	&#64;	<b>@</b>	96	60	140	&#96;	<b>`</b>
1	1	001	<b>SOH</b> (start of heading)	33	21	041	&#33;	<b>!</b>	65	41	101	&#65;	<b>A</b>	97	61	141	&#97;	<b>a</b>
2	2	002	<b>STX</b> (start of text)	34	22	042	&#34;	<b>"</b>	66	42	102	&#66;	<b>B</b>	98	62	142	&#98;	<b>b</b>
3	3	003	<b>ETX</b> (end of text)	35	23	043	&#35;	<b>#</b>	67	43	103	&#67;	<b>C</b>	99	63	143	&#99;	<b>c</b>
4	4	004	<b>EOT</b> (end of transmission)	36	24	044	&#36;	<b>\$</b>	68	44	104	&#68;	<b>D</b>	100	64	144	&#100;	<b>d</b>
5	5	005	<b>ENQ</b> (enquiry)	37	25	045	&#37;	<b>%</b>	69	45	105	&#69;	<b>E</b>	101	65	145	&#101;	<b>e</b>
6	6	006	<b>ACK</b> (acknowledge)	38	26	046	&#38;	<b>&amp;</b>	70	46	106	&#70;	<b>F</b>	102	66	146	&#102;	<b>f</b>
7	7	007	<b>BEL</b> (bell)	39	27	047	&#39;	<b>'</b>	71	47	107	&#71;	<b>G</b>	103	67	147	&#103;	<b>g</b>
8	8	010	<b>BS</b> (backspace)	40	28	050	&#40;	<b>(</b>	72	48	110	&#72;	<b>H</b>	104	68	150	&#104;	<b>h</b>
9	9	011	<b>TAB</b> (horizontal tab)	41	29	051	&#41;	<b>)</b>	73	49	111	&#73;	<b>I</b>	105	69	151	&#105;	<b>i</b>
10	A	012	<b>LF</b> (NL line feed, new line)	42	2A	052	&#42;	<b>*</b>	74	4A	112	&#74;	<b>J</b>	106	6A	152	&#106;	<b>j</b>
11	B	013	<b>VT</b> (vertical tab)	43	2B	053	&#43;	<b>+</b>	75	4B	113	&#75;	<b>K</b>	107	6B	153	&#107;	<b>k</b>
12	C	014	<b>FF</b> (NP form feed, new page)	44	2C	054	&#44;	<b>,</b>	76	4C	114	&#76;	<b>L</b>	108	6C	154	&#108;	<b>l</b>
13	D	015	<b>CR</b> (carriage return)	45	2D	055	&#45;	<b>-</b>	77	4D	115	&#77;	<b>M</b>	109	6D	155	&#109;	<b>m</b>
14	E	016	<b>SO</b> (shift out)	46	2E	056	&#46;	<b>.</b>	78	4E	116	&#78;	<b>N</b>	110	6E	156	&#110;	<b>n</b>
15	F	017	<b>SI</b> (shift in)	47	2F	057	&#47;	<b>/</b>	79	4F	117	&#79;	<b>O</b>	111	6F	157	&#111;	<b>o</b>
16	10	020	<b>DLE</b> (data link escape)	48	30	060	&#48;	<b>0</b>	80	50	120	&#80;	<b>P</b>	112	70	160	&#112;	<b>p</b>
17	11	021	<b>DC1</b> (device control 1)	49	31	061	&#49;	<b>1</b>	81	51	121	&#81;	<b>Q</b>	113	71	161	&#113;	<b>q</b>
18	12	022	<b>DC2</b> (device control 2)	50	32	062	&#50;	<b>2</b>	82	52	122	&#82;	<b>R</b>	114	72	162	&#114;	<b>r</b>
19	13	023	<b>DC3</b> (device control 3)	51	33	063	&#51;	<b>3</b>	83	53	123	&#83;	<b>S</b>	115	73	163	&#115;	<b>s</b>
20	14	024	<b>DC4</b> (device control 4)	52	34	064	&#52;	<b>4</b>	84	54	124	&#84;	<b>T</b>	116	74	164	&#116;	<b>t</b>
21	15	025	<b>NAK</b> (negative acknowledge)	53	35	065	&#53;	<b>5</b>	85	55	125	&#85;	<b>U</b>	117	75	165	&#117;	<b>u</b>
22	16	026	<b>SYN</b> (synchronous idle)	54	36	066	&#54;	<b>6</b>	86	56	126	&#86;	<b>V</b>	118	76	166	&#118;	<b>v</b>
23	17	027	<b>ETB</b> (end of trans. block)	55	37	067	&#55;	<b>7</b>	87	57	127	&#87;	<b>W</b>	119	77	167	&#119;	<b>w</b>
24	18	030	<b>CAN</b> (cancel)	56	38	070	&#56;	<b>8</b>	88	58	130	&#88;	<b>X</b>	120	78	170	&#120;	<b>x</b>
25	19	031	<b>EM</b> (end of medium)	57	39	071	&#57;	<b>9</b>	89	59	131	&#89;	<b>Y</b>	121	79	171	&#121;	<b>y</b>
26	1A	032	<b>SUB</b> (substitute)	58	3A	072	&#58;	<b>:</b>	90	5A	132	&#90;	<b>Z</b>	122	7A	172	&#122;	<b>z</b>
27	1B	033	<b>ESC</b> (escape)	59	3B	073	&#59;	<b>;</b>	91	5B	133	&#91;	<b>[</b>	123	7B	173	&#123;	<b>{</b>
28	1C	034	<b>FS</b> (file separator)	60	3C	074	&#60;	<b>&lt;</b>	92	5C	134	&#92;	<b>\</b>	124	7C	174	&#124;	<b> </b>
29	1D	035	<b>GS</b> (group separator)	61	3D	075	&#61;	<b>=</b>	93	5D	135	&#93;	<b>]</b>	125	7D	175	&#125;	<b>}</b>
30	1E	036	<b>RS</b> (record separator)	62	3E	076	&#62;	<b>&gt;</b>	94	5E	136	&#94;	<b>^</b>	126	7E	176	&#126;	<b>~</b>
31	1F	037	<b>US</b> (unit separator)	63	3F	077	&#63;	<b>?</b>	95	5F	137	&#95;	<b>_</b>	127	7F	177	&#127;	<b>DEL</b>

Source: [www.LookupTables.com](http://www.LookupTables.com)

TRANSFER				Flags								
Name	Comment	Code	Operation	O	D	I	T	S	Z	A	P	C
MOV	Move (copy)	MOV Dest,Source	Dest:=Source									
XCHG	Exchange	XCHG Op1,Op2	Op1:=Op2 , Op2:=Op1									
STC	Set Carry	STC	CF:=1									1
CLC	Clear Carry	CLC	CF:=0									0
CMC	Complement Carry	CMC	CF:= ¬,CF									±
STD	Set Direction	STD	DF:=1 (string op's downwards)		1							
CLD	Clear Direction	CLD	DF:=0 (string op's upwards)		0							
STI	Set Interrupt	STI	IF:=1			1						
CLI	Clear Interrupt	CLI	IF:=0			0						
PUSH	Push onto stack	PUSH Source	DEC SP, [SP]:=Source									
PUSHF	Push flags	PUSHF	O, D, I, T, S, Z, A, P, C 286+: also NT, IOPL									
PUSHA	Push all general registers	PUSHA	AX, CX, DX, BX, SP, BP, SI, DI									
POP	Pop from stack	POP Dest	Dest:=[SP], INC SP									
POPF	Pop flags	POPF	O, D, I, T, S, Z, A, P, C 286+: also NT, IOPL	±	±	±	±	±	±	±	±	±
POPA	Pop all general registers	POPA	DI, SI, BP, SP, BX, DX, CX, AX									
CBW	Convert byte to word	CBW	AX:=AL (signed)									
CWD	Convert word to double	CWD	DX:AX:=AX (signed)	±					±	±	±	±
CWDE	Conv word extended double	CWDE 386	EAX:=AX (signed)									
IN <i>i</i>	Input	IN Dest, Port	AL/AX/EAX := byte/word/double of specified port									
OUT <i>i</i>	Output	OUT Port, Source	Byte/word/double of specified port := AL/AX/EAX									



*i* for more information see instruction specifications

Flags: ±=affected by this instruction ?=undefined after this instruction

ARITHMETIC				Flags									
Name	Comment	Code	Operation	O	D	I	T	S	Z	A	P	C	
ADD	Add	ADD Dest,Source	Dest:=Dest+Source	±				±	±	±	±	±	
ADC	Add with Carry	ADC Dest,Source	Dest:=Dest+Source+CF	±				±	±	±	±	±	
SUB	Subtract	SUB Dest,Source	Dest:=Dest-Source	±				±	±	±	±	±	
SBB	Subtract with borrow	SBB Dest,Source	Dest:=Dest-(Source+CF)	±				±	±	±	±	±	
DIV	Divide (unsigned)	DIV Op	Op=byte: AL:=AX / Op AH:=Rest	?				?	?	?	?	?	
DIV	Divide (unsigned)	DIV Op	Op=word: AX:=DX:AX / Op DX:=Rest	?				?	?	?	?	?	
DIV 386	Divide (unsigned)	DIV Op	Op=doublew.: EAX:=EDX:EAX / Op EDX:=Rest	?				?	?	?	?	?	
IDIV	Signed Integer Divide	IDIV Op	Op=byte: AL:=AX / Op AH:=Rest	?				?	?	?	?	?	
IDIV	Signed Integer Divide	IDIV Op	Op=word: AX:=DX:AX / Op DX:=Rest	?				?	?	?	?	?	
IDIV 386	Signed Integer Divide	IDIV Op	Op=doublew.: EAX:=EDX:EAX / Op EDX:=Rest	?				?	?	?	?	?	
MUL	Multiply (unsigned)	MUL Op	Op=byte: AX:=AL*Op if AH=0 ♦	±				?	?	?	?	±	
MUL	Multiply (unsigned)	MUL Op	Op=word: DX:AX:=AX*Op if DX=0 ♦	±				?	?	?	?	±	
MUL 386	Multiply (unsigned)	MUL Op	Op=double: EDX:EAX:=EAX*Op if EDX=0 ♦	±				?	?	?	?	±	
IMUL i	Signed Integer Multiply	IMUL Op	Op=byte: AX:=AL*Op if AL sufficient ♦	±				?	?	?	?	±	
IMUL	Signed Integer Multiply	IMUL Op	Op=word: DX:AX:=AX*Op if AX sufficient ♦	±				?	?	?	?	±	
IMUL 386	Signed Integer Multiply	IMUL Op	Op=double: EDX:EAX:=EAX*Op if EAX sufficient ♦	±				?	?	?	?	±	
INC	Increment	INC Op	Op:=Op+1 (Carry not affected !)	±				±	±	±	±		
DEC	Decrement	DEC Op	Op:=Op-1 (Carry not affected !)	±				±	±	±	±		
CMP	Compare	CMP Op1,Op2	Op1-Op2	±				±	±	±	±	±	
SAL	Shift arithmetic left (=SHL)	SAL Op,Quantity		i				±	±	?	±	±	
SAR	Shift arithmetic right	SAR Op,Quantity		i				±	±	?	±	±	
RCL	Rotate left through Carry	RCL Op,Quantity		i								±	
RCR	Rotate right through Carry	RCR Op,Quantity		i								±	
ROL	Rotate left	ROL Op,Quantity		i								±	
ROR	Rotate right	ROR Op,Quantity		i								±	

*i* for more information see instruction specifications

♦ then CF:=0, OF:=0 else CF:=1, OF:=1

LOGIC				Flags									
Name	Comment	Code	Operation	O	D	I	T	S	Z	A	P	C	
NEG	Negate (two-complement)	NEG Op	Op:=0-Op if Op=0 then CF:=0 else CF:=1	±				±	±	±	±	±	
NOT	Invert each bit	NOT Op	Op:=¬Op (invert each bit)										
AND	Logical and	AND Dest,Source	Dest:=Dest∧Source	0				±	±	?	±	0	
OR	Logical or	OR Dest,Source	Dest:=Dest∨Source	0				±	±	?	±	0	
XOR	Logical exclusive or	XOR Dest,Source	Dest:=Dest (exor) Source	0				±	±	?	±	0	
SHL	Shift logical left (=SAL)	SHL Op,Quantity		<i>i</i>				±	±	?	±	±	
SHR	Shift logical right	SHR Op,Quantity		<i>i</i>				±	±	?	±	±	

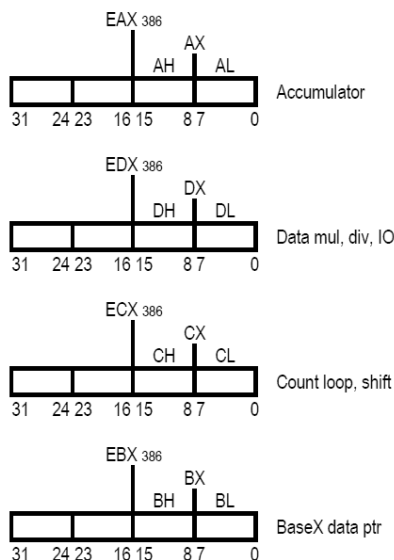


MISC				Flags								
Name	Comment	Code	Operation	O	D	I	T	S	Z	A	P	C
NOP	No operation	NOP	No operation									
LEA	Load effective address	LEA Dest,Source	Dest := address of Source									
INT	Interrupt	INT Nr	interrupts current program, runs spec. int-program			0	0					

JUMPS (flags remain unchanged)				Name	Comment	Code	Operation
Name	Comment	Code	Operation	Name	Comment	Code	Operation
CALL	Call subroutine	CALL Proc		RET	Return from subroutine	RET	
JMP	Jump	JMP Dest					
JE	Jump if Equal	JE Dest	(= JZ)	JNE	Jump if not Equal	JNE Dest	(= JNZ)
JZ	Jump if Zero	JZ Dest	(= JE)	JNZ	Jump if not Zero	JNZ Dest	(= JNE)
JCXZ	Jump if CX Zero	JCXZ Dest		JECXZ	Jump if ECX Zero	JECXZ Dest	<sup>386</sup>
JP	Jump if Parity (Parity Even)	JP Dest	(= JPE)	JNP	Jump if no Parity (Parity Odd)	JNP Dest	(= JPO)
JPE	Jump if Parity Even	JPE Dest	(= JP)	JPO	Jump if Parity Odd	JPO Dest	(= JNP)

JUMPS Unsigned (Cardinal)				JUMPS Signed (Integer)			
Name	Comment	Code	Operation	Name	Comment	Code	Operation
JA	Jump if Above	JA Dest	(= JNBE)	JG	Jump if Greater	JG Dest	(= JNLE)
JAE	Jump if Above or Equal	JAE Dest	(= JNB = JNC)	JGE	Jump if Greater or Equal	JGE Dest	(= JNL)
JB	Jump if Below	JB Dest	(= JNAE = JC)	JL	Jump if Less	JL Dest	(= JNGE)
JBE	Jump if Below or Equal	JBE Dest	(= JNA)	JLE	Jump if Less or Equal	JLE Dest	(= JNG)
JNA	Jump if not Above	JNA Dest	(= JBE)	JNG	Jump if not Greater	JNG Dest	(= JLE)
JNAE	Jump if not Above or Equal	JNAE Dest	(= JB = JC)	JNGE	Jump if not Greater or Equal	JNGE Dest	(= JL)
JNB	Jump if not Below	JNB Dest	(= JAE = JNC)	JNL	Jump if not Less	JNL Dest	(= JGE)
JNBE	Jump if not Below or Equal	JNBE Dest	(= JA)	JNLE	Jump if not Less or Equal	JNLE Dest	(= JG)
JC	Jump if Carry	JC Dest		JO	Jump if Overflow	JO Dest	
JNC	Jump if no Carry	JNC Dest		JNO	Jump if no Overflow	JNO Dest	
				JS	Jump if Sign (= negative)	JS Dest	
				JNS	Jump if no Sign (= positive)	JNS Dest	

## General Registers:

Flags: 

-	-	-	-	O	D	I	T	S	Z	-	A	-	P	-	C
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

## Control Flags (how instructions are carried out):

D: Direction 1 = string op's process down from high to low address  
I: Interrupt whether interrupts can occur. 1 = enabled  
T: Trap single step for debugging

## Example:

```

.DOSSEG           ; Demo program
.MODEL SMALL
.STACK 1024

Two EQU 2         ; Const
.DATA
VarB DB ?         ; define Byte, any value
VarW DW 1010b     ; define Word, binary
VarW2 DW 257      ; define Word, decimal
VarD DD 0AFFFFh   ; define Doubleword, hex
S DB "Hello!",0   ; define String
.CODE
main: MOV AX,DGROUP ; resolved by linker
      MOV DS,AX     ; init datasegment reg
      MOV [VarB],42 ; init VarB
      MOV [VarD],-7 ; set VarD
      MOV BX,Offset[S] ; addr of "H" of "Hello !"
      MOV AX,[VarW]  ; get value into accumulator
      ADD AX,[VarW2] ; add VarW2 to AX
      MOV [VarW2],AX ; store AX in VarW2
      MOV AX,4C00h   ; back to system
      INT 21h
      END main

```



## Status Flags (result of operations):

C: Carry result of unsigned op. is too large or below zero. 1 = carry/borrow  
O: Overflow result of signed op. is too large or small. 1 = overflow/underflow  
S: Sign sign of result. Reasonable for Integer only. 1 = neg. / 0 = pos.  
Z: Zero result of operation is zero. 1 = zero  
A: Aux. carry similar to Carry but restricted to the low nibble only  
P: Parity 1 = result has even number of set bits

## Vedlegg - Linux/unix systemkall

%eax	Name	%ebx	%ecx	%edx	%esx	%edi
1	sys_exit	int	-	-	-	-
2	sys_fork	struct pt_regs	-	-	-	-
3	sys_read	unsigned int	char *	size_t	-	-
4	sys_write	unsigned int	const char *	size_t	-	-
5	sys_open	const char *	int	int	-	-
6	sys_close	unsigned int	-	-	-	-