

HØGSKOLEN  
I BERGEN

# HØGSKOLEN I BERGEN

Avdeling for ingeniørutdanning  
Institutt for data- og realfag

---

EKSAMEN I : DAT103 Operativsystemer

KLASSE : DATA/INF

DATO : 15/12/16

---

ANTALL OPPGAVER : 6  
ANTAL SIDER : 4, vedlegg er ikke inkludert  
VEDLEGG : ascii-tabell,  
intel x86 kodetabell,  
linux systemkall

HJELPEMIDLER : Ingen

TID : 4 timer

MÅLFORM : Bokmål

SENSOR :

FAGLÆRERE : Pål Ellingsen, Atle Geitung

## Oppgave 1 – bash (15% ~ 36 minutter)

Gitt skall-programmet til høyre:

- a) Forklar hva hver enkelt linje gjør i det gitte programmet.
- b) Gi en overordnet forklaring på hva det gitte programmet gjør.
- c) Koden legges inn i en tekstfil med navnet program.sh. Hva må vi gjøre for å kunne kjøre programmet fra kommandolinjen?
- d) Lag et skall-program som først skriver ut navnet på alle kjørbare (executable) filene i inneværende katalog og deretter antallet kjørbare filer. Hvis du ikke husker helt kommandonavn, kommando-opsjoner eller resultatet av en kommando, gjør antakelser som du skriver ned.

```
1  #!/bin/bash
2
3  if [ $# -ne 1 ]
4  then
5      echo 'Feil'
6      exit 1
7  fi
8
9  result=1
10 n=$1
11 while (( n >= 1 ))
12 do
13     (( result = n * result ))
14     (( n = n - 1 ))
15 done
16 echo $result
```

## Oppgave 2 – Operativsystem (10% ~ 24 minutter)

- a) Hva er et operativsystem?
- b) Forklar hvordan operativsystemet kan organisere filene i sekundærlageret.

## Oppgave 3 (20% ~ 48 minutter)

Vi skal se på 5 prosesser P0, P1, P2, P3 og P4. Disse prosessene skal bruke ressursene A, B, C og D.

På et punkt i utførelsen av disse prosessene har vi følgende tilstand:

	Allokert	Maks behov	Tilgjengelig
	A B C D	A B C D	A B C D
P0	2 0 0 1	4 2 1 2	3 3 2 1
P1	3 1 2 1	5 2 5 2	
P2	2 1 0 3	2 3 1 6	
P3	1 3 1 2	1 4 2 4	
P4	1 4 3 2	3 6 6 5	

- a) Finn ut om systemet er i en sikker tilstand eller ikke. Begrunn svaret ditt.
- b) Kan forespørselen (1 1 0 0) fra P1 godkjennes?
- c) Kan forespørselen (0 0 2 0) fra P2 godkjennes?
- d) Det er 4 nødvendige betingelser som må være til stede for at en vranglås skal kunne oppstå.  
List opp disse betingelsene og forklar hvordan de bidrar til vranglås

#### Oppgave 4 (10% ~ 24 minutter)

Anta at vi har 5 prosesser P0, P1, P2, P3 og P4 med følgende CPU-utbruddstider (burst time):

Prosess	Utbruddstid	Ankomsttid
P0	10	0
P1	7	1
P2	3	2
P3	5	3
P4	1	4

Vis med diagram eller på annen passende måte hvordan disse prosessene blir kjørt med følgende algoritmer, og beregn gjennomsnittlig ventetid for prosessene:

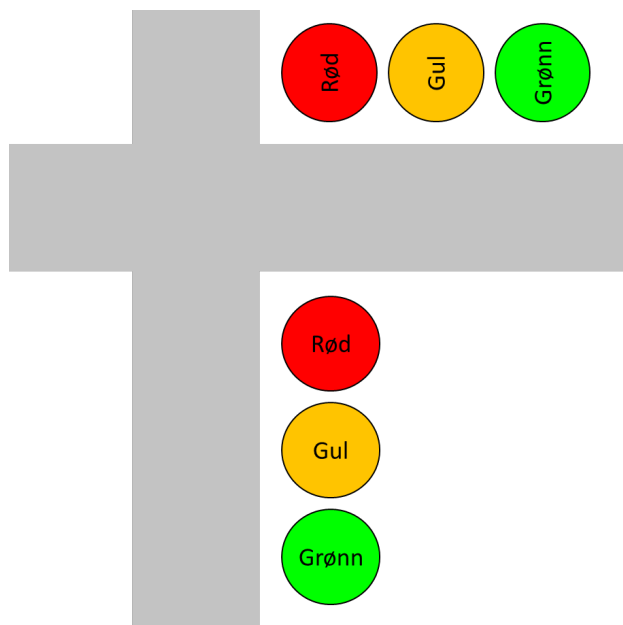
- a) First-Come, First-Served
- b) Shortest-Job-First
- c) Round Robin med lukestørrelse 2
- d) Round Robin med lukestørrelse 3

Forklar hvorfor gjennomsnittlig omløpstid med Round Robin algoritmen (turnaround time) ikke nødvendigvis øker med lukestørrelsen.

#### Oppgave 5 – Digitalteknikk (25% ~ 60 minutter)

Du skal lage et styringssystem (dekoder) for et trafikkllys i et veikryss. Se figuren for et veikryss hvor to veier krysser hverandre, en i retning Øst-Vest og en i retning Sør-Nord. I figuren er det kun tegnet inn ett lys i hver retning da det på motsatt side vil ha tilsvarende tilstand. Merk at lysene fra nord og fra sør alltid lyser likt og det samme gjelder lysene fra øst og fra vest. Trafikkllyset i dette krysset har normalt følgende sekvens:

Sør-Nord	Øst-Vest	Hurtig sekvens
Rød	Grønn	Nei
Rød/Gul	Gul	Nei
Rød	Rød	Nei
Grønn	Rød	Nei
Gul	Rød/Gul	Nei
Rød	Rød	Nei



I tillegg til standardsekvensen, skal også lyset ha en hurtigere overgang mellom fasene. Denne kan benyttes når det er mindre trafikk i krysset. Da er sekvensen som følger:

Sør-Nord	Øst-Vest	Hurtig sekvens
Rød	Grønn	Ja
Rød	Gul	Ja
Grønn	Rød	Ja
Gul	Rød	Ja

Kall lysene i retning Sør-Nord for a (grønn), b (gul) og c (rød) og lysene i retning Øst-Vest for d (grønn), e (gul) og f (rød).

Hurtigtilstand skal vises med en liten indikator. Denne kaller du g.

For å kunne ha 10 tilstander, må du ha 4 verdier (bit) inn. Kall dem ABCD.

- Sett opp sannhetstabellen for hele dekoderen (for både a, b, c, d, e, og f).
- Sett opp karnaughdiagram for hele dekoderen. Det enkleste er å ha ett diagram for hvert av lysene a til f.
- Finn enklest mulig boolsk uttrykk for dekoderen.
- Tegn kretsløsningen for dekoderen (de boolske uttrykkene fra c)).

## Oppgave 6 – Assembly (20% ~ 48 minutter)

Til høyre er assemblykoden til et program listet.

- Forklar koden i detalj.
- Forklar hva programmet gjør (en overordnet forklaring).
- Oversett følgende Java- algoritme til assembly-kode.

```

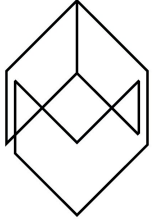
1  /**
2   * Skriver ut de første n tallene i
3   * en fibonacci-følge.
4   *
5   * @param n antall fibonacci-tall
6   */
7  public void printFib(int n) {
8      int a = 1;
9      int b = 2;
10
11     for (int i = n; i != 0; i--) {
12         System.out.println(a);
13         int t = a;
14         a = b;
15         b = t + b;
16     }
17 }

```

```

1  section .bss
2      n resb 1
3
4  section .text
5      global _start
6  _start:
7      mov ecx, 10
8      mov eax, '1'
9
10 a:
11     mov [n], eax
12     mov eax, 4
13     mov ebx, 1
14     push ecx
15
16     mov ecx, n
17     mov edx, 1
18     int 0x80
19
20     mov eax, [n]
21     sub eax, '0'
22     inc eax
23     add eax, '0'
24     pop ecx
25     loop a
26
27     mov eax, 1
28     int 0x80

```



HØGSKOLEN  
I BERGEN

# HØGSKOLEN I BERGEN

Avdeling for ingeniørutdanning  
Institutt for data- og realfag

---

EKSAMEN I : DAT103 Operativsystem

KLASSE : DATA/INF

DATO : 15/12/16

---

TAL PÅ OPPGÅVER : 6  
TAL PÅ SIDER : 4, vedlegg er ikkje inkludert  
VEDLEGG : ascii-tabell,  
intel x86 kodetabell,  
linux systemkall

HJELPEMIDDEL : Ingen

TID : 4 timer

MÅLFORM : Nynorsk

SENSOR :

FAGLÆRARE : Pål Ellingsen, Atle Geitung

## Oppgåve 1 – bash (15% ~ 36 minutt)

Gjeven skal-programmet til høgre:

- a) Forklar kva kvar einskild line gjer i det gjevne programmet.
- b) Gje ei overordna forklaring på kva det gjevne programmet gjer.
- c) Koden leggas inn i ei tekstfil med namnet program.sh. Kva må vi gjera for å kunne køyre programmet frå kommandolina?
- d) Lag eit skal-program som først skriv ut namna på alle køyrbare (executable) filene i inneverande katalog og deretter talet på køyrbare filer. Viss du ikkje hugsar heilt kommandonamn, kommando-opsjonar eller resultat av ei kommando, gjer føresetnader som du skriv ned.

```
1  #!/bin/bash
2
3  if [ $# -ne 1 ]
4  then
5      echo 'Feil'
6      exit 1
7  fi
8
9  result=1
10 n=$1
11 while (( n >= 1 ))
12 do
13     (( result = n * result ))
14     (( n = n - 1 ))
15 done
16 echo $result
```

## Oppgåve 2 – Operativsystem (10% ~ 24 minutt)

- a) Kva er eit operativsystem?
- b) Forklar korleis operativsystemet kan organisere filene i sekundærlageret.

## Oppgåve 3 (20% ~ 48 minutt)

Vi skal sjå på 5 prosessar P0, P1, P2, P3 og P4. Disse prosessane skal bruke ressursane A, B, C og D.

På eit punkt i utføringa av disse prosessane har vi følgjande tilstand:

	Allokert	Maks behov	Tilgjengelig
	A B C D	A B C D	A B C D
P0	2 0 0 1	4 2 1 2	3 3 2 1
P1	3 1 2 1	5 2 5 2	
P2	2 1 0 3	2 3 1 6	
P3	1 3 1 2	1 4 2 4	
P4	1 4 3 2	3 6 6 5	

- a) Finn ut om systemet er i ein sikker tilstand eller ikkje. Begrunn svaret ditt.
- b) Kan forespørselen (1 1 0 0) frå P1 godkjennast?
- c) Kan forespørselen (0 0 2 0) frå P2 godkjennast?
- d) Det er 4 nødsynte føresetnader som må vere til stade for at ein vranglås skal kunne oppstå.  
List opp disse føresetnadene og forklar korleis dei bidrar til vranglås.

#### Oppgåve 4 (10% ~ 24 minutt)

Anta at vi har 5 prosessar P0, P1, P2, P3 og P4 med følgjande CPU-utbruddstider (burst time):

Prosess	Utbruddstid	Ankomsttid
P0	10	0
P1	7	1
P2	3	2
P3	5	3
P4	1	4

Vis med diagram eller på anna passande måte korleis disse prosessane blir kjørt med følgjande algoritmer, og berekn gjennomsnittleg ventetid for prosessane:

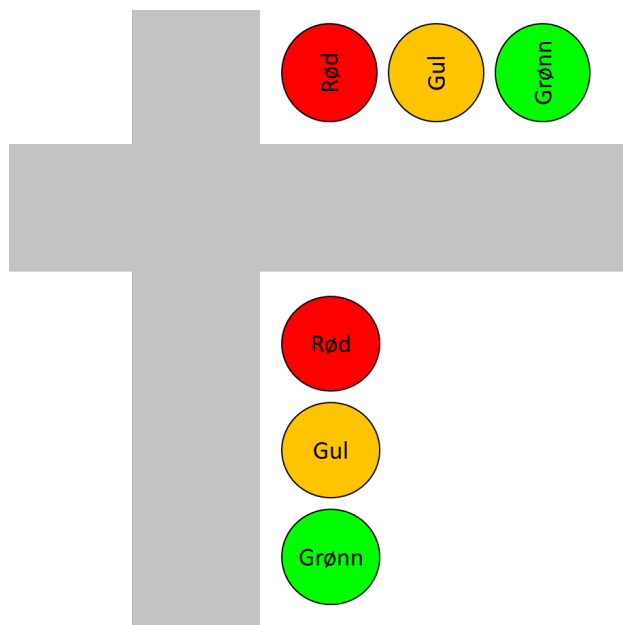
- a) First-Come, First-Served
- b) Shortest-Job-First
- c) Round Robin med lukestorleik 2
- d) Round Robin med lukestorleik 3

Forklar korfor gjennomsnittleg omløpstid med Round Robin algoritmen (turnaround time) ikkje nødvendigvis aukar med lukestorleiken.

#### Oppgåve 5 – Digitalteknikk (25% ~ 60 minutt)

Du skal lage eit styringssystem (dekodar) for eit trafikklys i eit vegkryss. Sjå figuren for eit vegkryss kor to veger kryssar kvarandre, ein i retning Aust-Vest og ein i retning Sør-Nord. I figuren er det berre teikna inn eitt lys i kvar retning då det på motsett side vil ha tilsvarende tilstand. Merk at ljosa frå nord og frå sør alltid ljoser likt og det same gjeld ljosa frå aust og frå vest. Trafikklyset i dette krysset har normalt fylgjande sekvens:

Sør-Nord	Aust-Vest	Hurtig sekvens
Raud	Grøn	Nei
Raud/Gul	Gul	Nei
Raud	Raud	Nei
Grøn	Raud	Nei
Gul	Raud/Gul	Nei
Raud	Raud	Nei



I tillegg til standardsekvensen, skal og ljaset ha ein hurtigare overgang mellom fasane. Denne kan nyttas når det er mindre trafikk i krysset. Da er sekvensen som følger:

Sør-Nord	Aust-Vest	Hurtig sekvens
Raud	Grøn	Ja
Raud	Gul	Ja
Grøn	Raud	Ja
Gul	Raud	Ja

Kall ljosa i retning Sør-Nord for a (grøn), b (gul) og c (raud) og ljosa i retning Aust-Vest for d (grøn), e (gul) og f (raud).

Hurtigtilstand skal synast med ein liten indikator. Denne kallar du g.

For å kunne ha 10 tilstander, må du ha 4 verdier (bit) inn. Kall dei ABCD.

- Sett opp sanningstabellen for heile dekodaren (for både a, b, c, d, e, f og g).
- Sett opp karnaughdiagram for heile dekodaren. Det enklaste er å ha eitt diagram for kvart av ljosa a til g.
- Finn enklast mogeleg boolsk uttrykk for dekodaren.
- Teikn kretsløysninga for dekodaren (dei boolske uttrykka frå c)).

## Oppgave 6 – Assembly (20% ~ 48 minutt)

Til høgre er assemblykoden til eit program lista.

- Forklar koden i detalj.
- Forklar kva programmet gjer (ei overordna forklaring).
- Oversett følgjande Java-algoritme til assembly-kode.

```

1  /**
2   * Skriver ut de første n tallene i
3   * en fibonacci-følge.
4   *
5   * @param n antall fibonacci-tall
6   */
7  public void printFib(int n) {
8      int a = 1;
9      int b = 2;
10
11     for (int i = n; i != 0; i--) {
12         System.out.println(a);
13         int t = a;
14         a = b;
15         b = t + b;
16     }
17 }

```

```

1  section .bss
2      n resb 1
3
4  section .text
5      global _start
6  _start:
7      mov ecx, 10
8      mov eax, '1'
9
10 a:
11     mov [n], eax
12     mov eax, 4
13     mov ebx, 1
14     push ecx
15
16     mov ecx, n
17     mov edx, 1
18     int 0x80
19
20     mov eax, [n]
21     sub eax, '0'
22     inc eax
23     add eax, '0'
24     pop ecx
25     loop a
26
27     mov eax, 1
28     int 0x80

```



## 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									


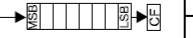
*i* for more information see instruction specifications

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

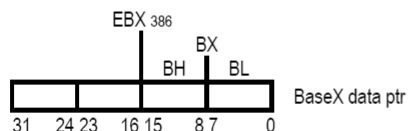
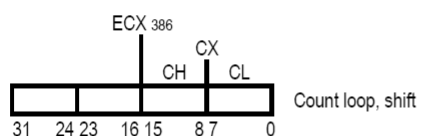
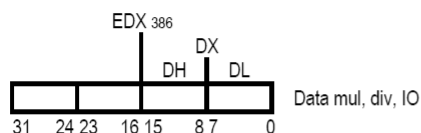
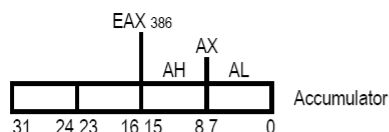
LOGIC				Flags								
Name	Comment	Code	Operation	O	D	I	T	S	Z	A	P	C
NEG	Negate (two-complement)	NEG Op	Op:=0-Op            if Op=0 then CF:=0 else CF:=1	±				±	±	±	±	±
NOT	Invert each bit	NOT Op	Op:=¬Op (invert each bit)									
AND	Logical and	AND Dest,Source	Dest:=Dest∧Source	0				±	±	?	±	0
OR	Logical 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
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	386
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)
JNBLe	Jump if not Below or Equal	JNBLe 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:



## 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

```

Flags: ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐

## 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

## 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	-	-	-	-