



Høgskulen på Vestlandet

Avdeling for ingeniørutdanning
Institutt for data- og realfag

EKSAMEN I : DAT103 Datamaskiner og operativsystem

KLASSE : DATA/INF

DATO : 8. juni 2017

ANTAL OPPGÅVER : 6

ANTAL SIDER : 8 (inkludert forside og vedlegg)

VEDLEGG : 3 vedlegg på sidene 6-8.
ASCII-tabell, unix/linux systemkall
og x86-instruksjoner

HJELPEMIDDEL : Ingen

TID : 4 timar

MÅLFORM : Nynorsk

SENSOR :

FAGLÆRARAR : Pål Ellingsen,
Atle Geitung

KOMMENTAR :

Oppgåve 1 – bash (15% ~ 36 minutt)

Gitt skal-programmet til høgre:

- a) Forklar kva kvar enkelt linje gjer i det gitte programmet.
- b) Gje ei overordna forklaring på kva det gitte programmet gjer.
- c) Koden skal leggjast inn i ei tekstfil med namnet program.sh. Kva må vi gjere for å kunne køyre programmet frå kommandolinja?
- d) Lag eit skal-program som tar ein parameter som er eit filnamn eller ein del av et filnamn. Programmet skal skrive ut alle filnamna som inneheld filnamn-delen som er gitt. Dersom du ikkje hugsar heilt kommandonamn, kommando-opisjonar eller resultatet av ein kommando, gjer naudsynte antakelser og skriv dei ned.

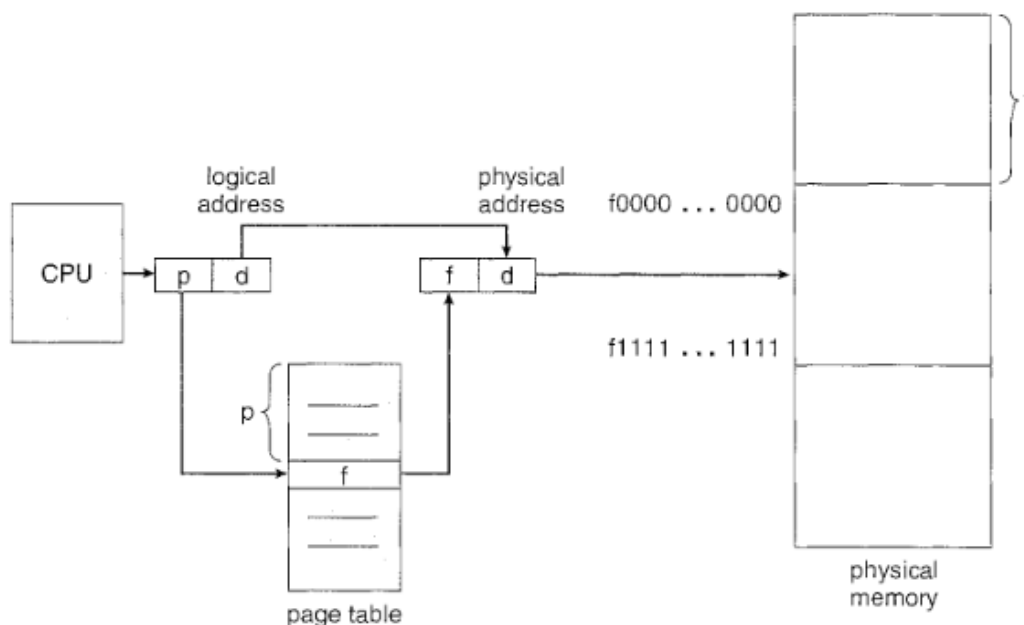
```
1  #!/bin/bash
2
3  if [ $# -ne 1 ]
4  then
5      echo 'Feil'
6      exit 1
7  fi
8
9  result=1
10 n=$1
11 while (( n >= 1 ))
12 do
13     (( result = n * result ))
14     (( n = n - 1 ))
15 done
16 echo $result
```

Oppgåve 2 – Operativsystem (10% ~ 24 minutt)

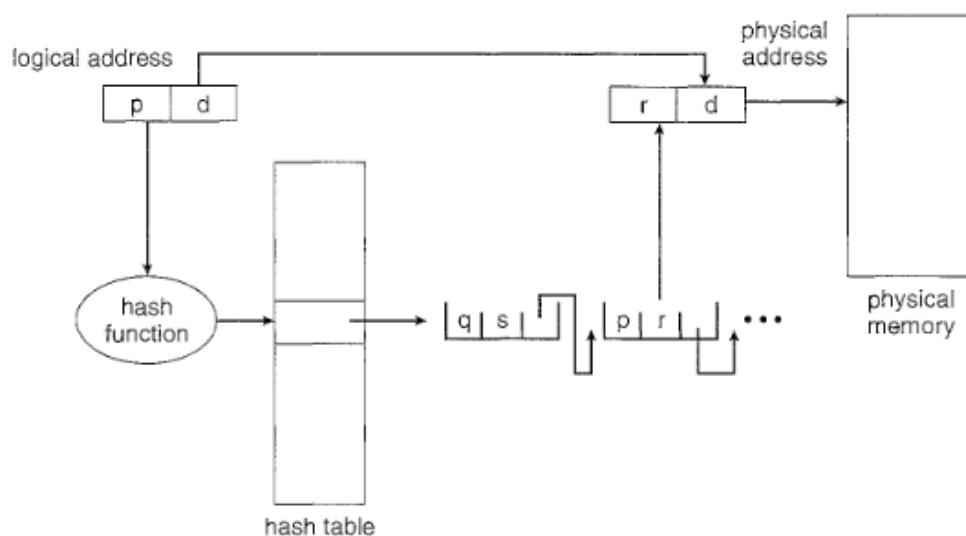
- a) Kva er eit operativsystem?
- b) Forklar korleis operativsystemet kan organisere filene i sekundærlageret (disken).

Oppgave 3 (20% ~ 48 minutt)

- a) I samband med minneallokering har vi tre hovedstrategiar for korleis ein best utnyttar ledige minneblokkar: First fit, Best fit og Worst fit. Forklar kva desse strategiane inneber.
- b) Forklar kva sidedeling (paging) er og korfor dette er ein god teknikk for minnehandtering.
- c) Forklar korleis sidedeling fungerer ut frå figuren under:

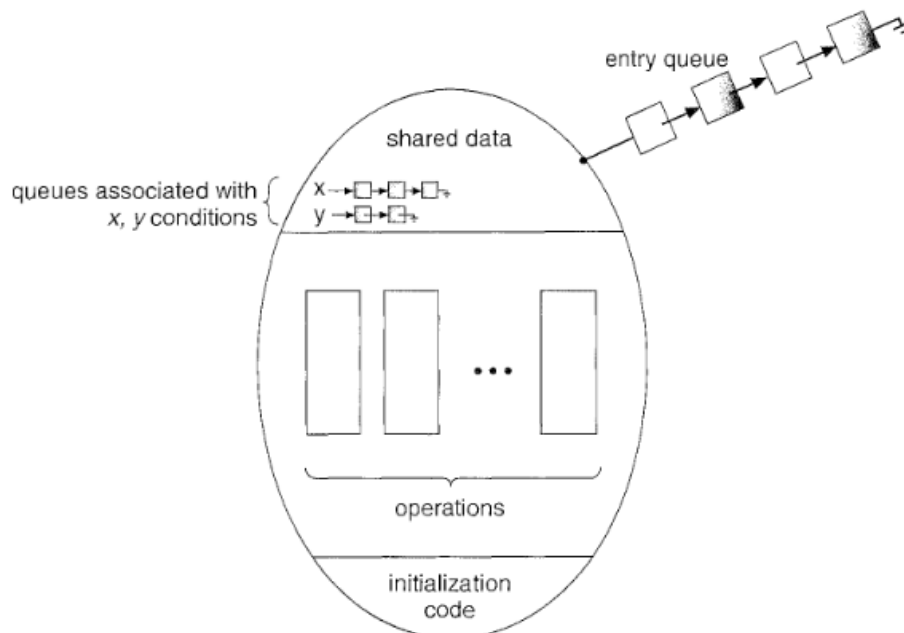


- d) Forklar kva ein sidetabell (page table) er.
- e) Forklar kva ein hashet sidetabell er og korfor ein bruker dette.
- f) Forklar korleis ein hashet sidetabell fungerer ut frå figuren under.



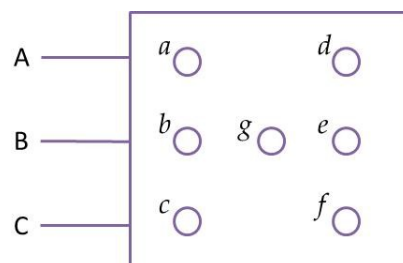
Oppgave 4 (10% ~ 24 minutter)

- Forklar korleis dei tre synkroniseringsmekanismane Mutex, Semafor og Monitor fungerer.
- Figuren under viser ein monitor med betingelsesvariable (condition variables). Forklar kordan betingelsesvariable fungerer ut frå figuren under.

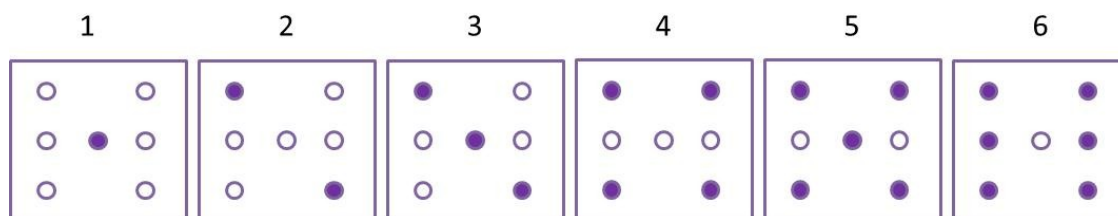


Oppgave 5 – Digitalteknikk (25% ~ 60 minut)

Figuren til høgre viser ein digital terning bestående av 7 dioder som blir styrt av ein dekode. Den har tre inngangar, A, B og C. Dekoderen skal dekode verdiane 1, 2, 3, 4, 5 og 6 slik at riktige dioder lyser og viser terningen sin verdi på same måte som ein vanleg terning, sjå figuren under for dekoding. Talet blir representert digitalt med sifrane CBA og andre verdier enn 1, 2, 3, 4, 5 og 6 vil ikkje forekomme.



For eksempel vil ein ener vere representert som CBA = 001 og dioden g i figuren vil lyse.



- Sett opp sannhetstabellen for heile dekodeeren (for både a, b, c, d, e, f og g).
- Sett opp karnaughdiagram for d, e, f og g. Det enklaste er å ha ett diagram for kvart av lysa.
- Finn eit enklast mogleg boolsk uttrykk for d, e, f og g.
- Teikn kretsløysinga for dekodeeren (dei boolske uttrykka frå c).

Oppgave 6 – Assembly (20% ~ 48 minutter)

Til høyre er assemblykoden til eit program lista.

- Forklar koden i detalj.
- Forklar kva programmet gjer (ei overordna forklaring).
- Oversett følgjande Java- algoritme til assembly-kode. Du kan anta at funksjonen skrivTall er tilgjengeleg og at talverdien som skal skrivast ut blir overført i eax til skrivTall.

```
1 private static final n = 5;
2
3 /**
4  * Summerer de første n naturlige tallene.
5  * (1 + 2 + ... + n)
6  */
7 public void sumNaturligeTall() {
8     int sum = 0;
9     for (int i = 1; i <= n; i++) {
10         sum += i;
11     }
12     skrivTall(sum);
13 }
14
```

```
1 cr equ 13
2 lf equ 10
3
4 section .data
5 msg1 db 'Inndata: '
6 n1 equ $ - msg1
7 msg2 db cr,lf,'ja'
8 n2 equ $ - msg2
9 msg3 db cr,lf,'nei'
10 n3 equ $ - msg3
11
12 section .bss
13 a resb 1
14
15 section .text
16 global _start
17 _start:
18     mov eax, 4
19     mov ebx, 1
20     mov ecx, msg1
21     mov edx, n1
22     int 80h
23
24     mov eax, 3
25     mov ebx, 0
26     mov ecx, a
27     mov edx, 1
28     int 80h
29
30     mov eax, [a]
31     and eax, 1
32     cmp eax, 0
33     je _nei
34 _ja:
35     mov eax, 4
36     mov ebx, 1
37     mov ecx, msg2
38     mov edx, n2
39     int 80h
40     jmp _ferdig
41 _nei:
42     mov eax, 4
43     mov ebx, 1
44     mov ecx, msg3
45     mov edx, n3
46     int 80h
47 _ferdig:
48     mov eax, 1
49     mov ebx, 0
50     int 80h
51
```



Høgskulen på Vestlandet

Avdeling for ingeniørutdanning
Institutt for data- og realfag

EKSAMEN I : DAT103 Datamaskiner og operativsystemer

KLASSE : DATA/INF

DATO : 8. juni 2017

ANTALL OPPGAVER : 6

ANTAL SIDER : 8 (inkludert forside og vedlegg)

VEDLEGG : 3 vedlegg på sidene 6-8.
ASCII-tabell, unix/linux systemkall
og x86-instruksjoner

HJELPEMIDDEL : Ingen

TID : 4 timer

MÅLFORM : Bokmål

SENSOR :

FAGLÆRERE : Pål Ellingsen,
Atle Geitung

KOMMENTAR :

Oppgave 1 – bash (15% ~ 36 minutter)

Gitt skall-programmet til høyre:

e) Forklar hva hver enkelt linje gjør i det gitte programmet.

f) Gi en overordnet forklaring på hva det gitte programmet gjør.

g) Koden legges inn i en tekstfil med navnet program.sh. Hva må vi gjøre for å kunne kjøre programmet fra kommandolinjen?

h) Lag et skall-program som tar en parameter som er et filnavn eller en del av et filnavn. Programmet skal skrive ut alle filnavnene som inneholdt filnavn-delen som er gitt. Hvis du ikke husker helt kommandonavn, kommando-opsjoner eller resultatet av en kommando, gjør nødvendige antakelser og skriver dem 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
```

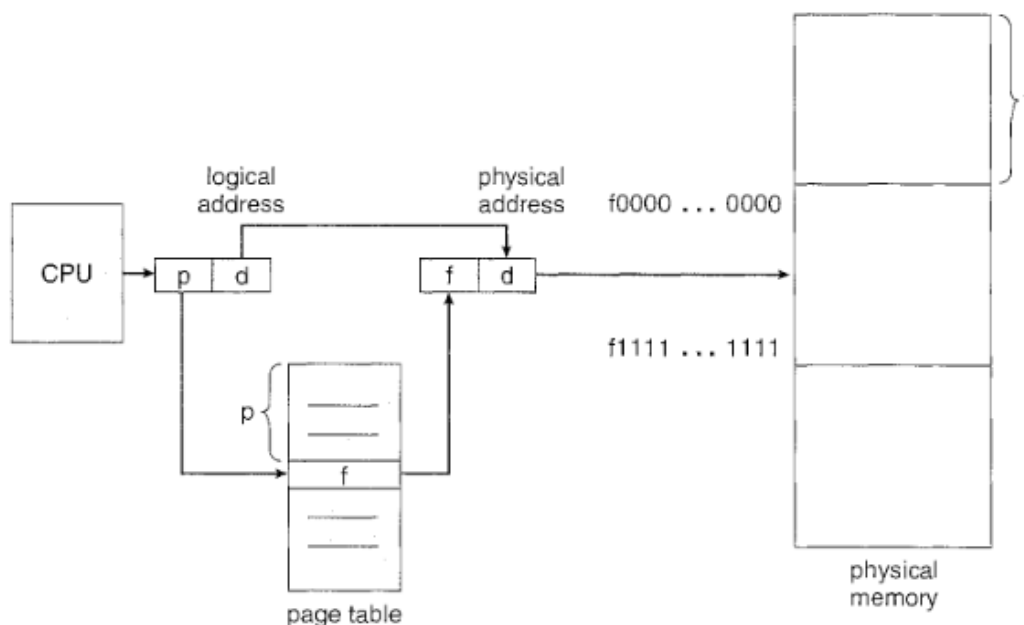
Oppgave 2 – Operativsystem (10% ~ 24 minutter)

c) Hva er et operativsystem?

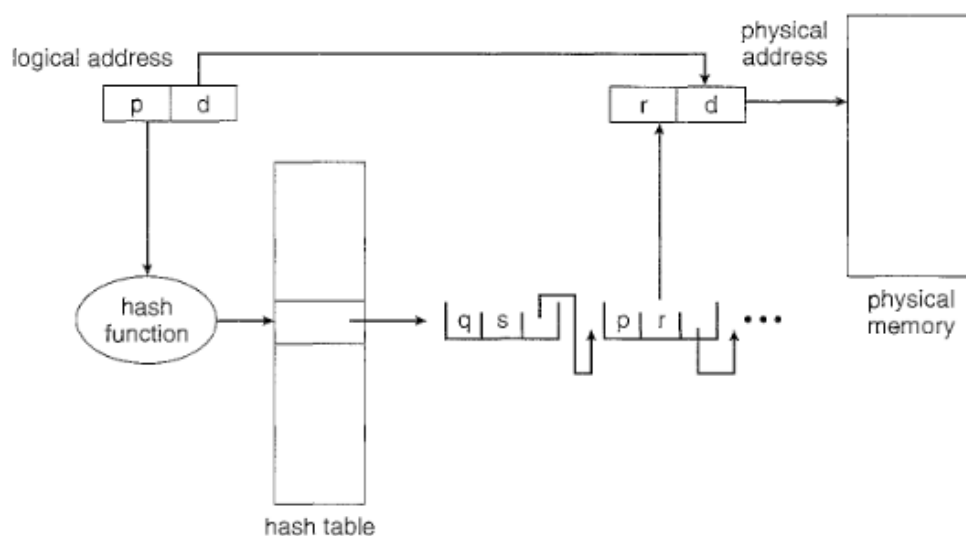
d) Forklar hvordan operativsystemet kan organisere filene i sekundærlageret (disken).

Oppgave 3 (20% ~ 48 minutter)

- g) I forbindelse med minneallokering har vi tre hovedstrategier for hvordan man best utnytter ledige minneblokker: First fit, Best fit og Worst fit. Forklar hva disse strategiene innebærer.
- h) Forklar hva sidedeling (paging) er og hvorfor dette er en god teknikk for minnehåndtering.
- i) Forklar hvordan sidedeling fungerer ut fra figuren under:

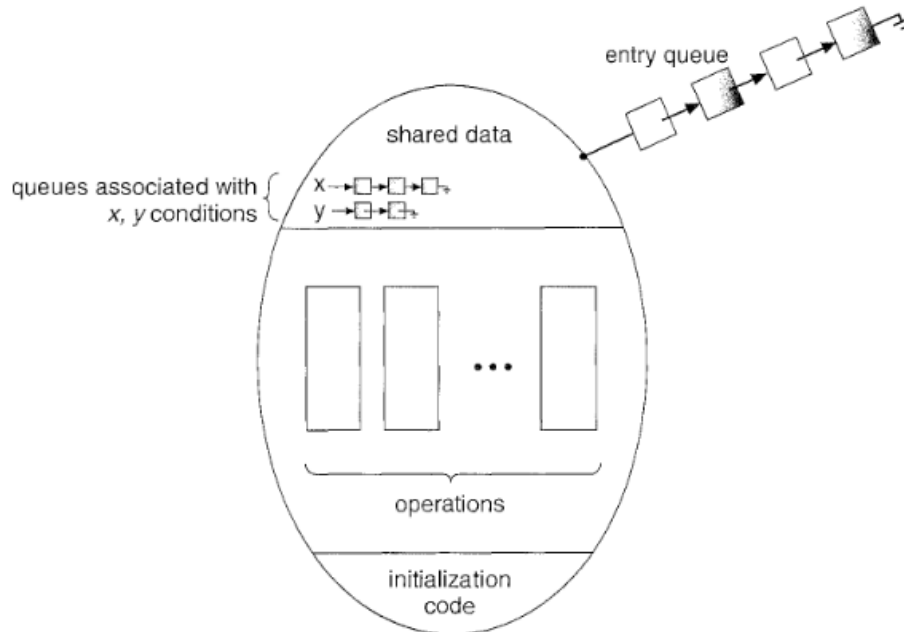


- j) Forklar hva en sidetabell (page table) er.
- k) Forklar hva en hashet sidetabell er og hvorfor man bruker dette.
- l) Forklar hvordan hashet sidetabell fungerer ut fra figuren under.



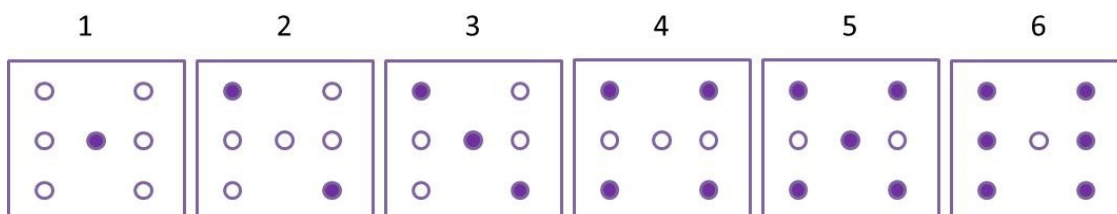
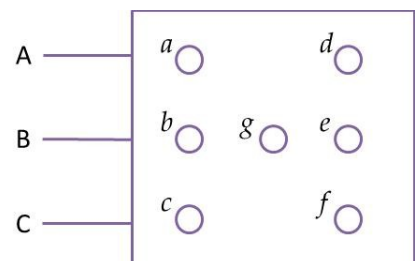
Oppgave 4 (10% ~ 24 minutter)

- c) Forklar hvordan de tre synkroniseringsmekanismene Mutex, Semafor og Monitor fungerer.
- d) Figuren under viser en monitor med betingelsesvariable (condition variables). Forklar hvordan betingelsesvariable fungerer ut fra figuren under.



Oppgave 5 – Digitalteknikk (25% ~ 60 minutter)

Figuren til høyre viser en digital terning bestående av 7 dioder som styres av en dekode. Den har tre innganger, A, B og C. Dekoderen skal dekode verdiene 1, 2, 3, 4, 5 og 6 slik at riktige dioder lyser og viser terningens verdi på samme måte som en vanlig terning, se figuren under for dekoding. Tallet representeres digital med sifrene CBA og andre verdier enn 1, 2, 3, 4, 5 og 6 vil ikke forekomme. For eksempel vil en ener være representert som CBA = 001 og dioden g i figuren vil lyse.



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

Oppgave 6 – Assembly (20% ~ 48 minutter)

Til høyre er assemblykoden til et program listet.

- d) Forklar koden i detalj.
- e) Forklar hva programmet gjør (en overordnet forklaring).
- f) Oversett følgende Java- algoritme til assembly-kode. Du kan anta at funksjonen skrivTall er tilgjengelig og at tallverdien som skal skrives ut overføres i eax til skrivTall.

```
1 private static final n = 5;
2
3 /**
4  * Summerer de første n naturlige tallene.
5  * (1 + 2 + ... + n)
6  */
7 public void sumNaturligeTall() {
8     int sum = 0;
9     for (int i = 1; i <= n; i++) {
10         sum += i;
11     }
12     skrivTall(sum);
13 }
14
```

```
1 cr equ 13
2 lf equ 10
3
4 section .data
5 msg1 db 'Inndata: '
6 n1 equ $ - msg1
7 msg2 db cr,lf,'ja'
8 n2 equ $ - msg2
9 msg3 db cr,lf,'nei'
10 n3 equ $ - msg3
11
12 section .bss
13 a resb 1
14
15 section .text
16 global _start
17 _start:
18     mov eax, 4
19     mov ebx, 1
20     mov ecx, msg1
21     mov edx, n1
22     int 80h
23
24     mov eax, 3
25     mov ebx, 0
26     mov ecx, a
27     mov edx, 1
28     int 80h
29
30     mov eax, [a]
31     and eax, 1
32     cmp eax, 0
33     je _nei
34 _ja:
35     mov eax, 4
36     mov ebx, 1
37     mov ecx, msg2
38     mov edx, n2
39     int 80h
40     jmp _ferdig
41 _nei:
42     mov eax, 4
43     mov ebx, 1
44     mov ecx, msg3
45     mov edx, n3
46     int 80h
47 _ferdig:
48     mov eax, 1
49     mov ebx, 0
50     int 80h
51
```

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

Source: www.LookupTables.com






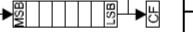
Vedlegg - Linux/unix systemkall

%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	-	-	-	-

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>i</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>i</i>					±	±	?	±
SAR	Shift arithmetic right	SAR Op,Quantity		<i>i</i>					±	±	?	±
RCL	Rotate left through Carry	RCL Op,Quantity		<i>i</i>								±
RCR	Rotate right through Carry	RCR Op,Quantity		<i>i</i>								±
ROL	Rotate left	ROL Op,Quantity		<i>i</i>								±
ROR	Rotate right	ROR Op,Quantity		<i>i</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 (xor) Source	0				±	±	?	±	0
SHL	Shift logical left (≡ SAL)	SHL Op,Quantity		<i>i</i>					±	±	?	±
SHR	Shift logical right	SHR Op,Quantity		<i>i</i>					±	±	?	±

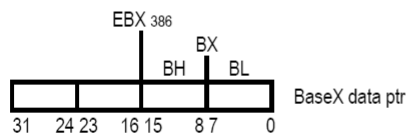
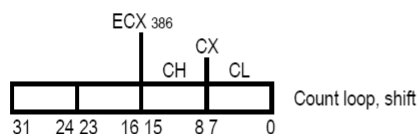
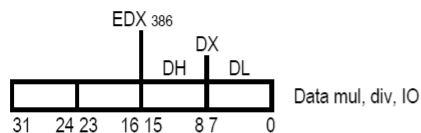
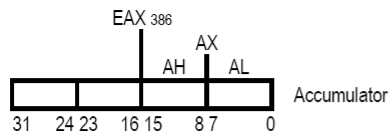
Download latest version free of charge from www.jegerlehner.ch/intel This page may be freely distributed without cost provided it is not changed. All rights reserved

MISC		Code	Operation	Flags								
Name	Comment			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	JAЕ 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:



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