Western Norway University of Applied Sciences

Department of Computer science, Electrical engineering and Mathematical sciences

Examination in:	DAT103 – Computers and Operating Systems
Day of examination:	13 June 2023
Time of examination:	9:00 – 13:00
Permitted aids:	Regular calculator
Language:	English

This problem set consists of 10 pages, excluding the appendix.
You can answer in either English or Norwegian.

Good Luck!
Dag and Violet

Problem 1 – Multiple Choice Questions (24%)

For each of the multiple choice questions below, select at most one choice.

1.1 Which of the following allow I/O modules and main memory exchange data directly?

(a) Memory management unit (MMU)
(b) Direct memory access (DMA)
(c) Memory address register
(d) All of the above
(e) None of the above

Solution: Direct memory access (DMA) □

1.2 Which of the following can explain why having more random access memory (RAM) can improve computer performance?

(a) Having more RAM can reduce external fragmentation
(b) Larger RAM can increase processor speed
(c) Having more RAM can reduce the possibility of page faults
(d) All of the above
(e) None of the above

Solution: Having more RAM can reduce the possibility of page faults □

1.3 Which of the following will be saved during a *context switch* between processes?

(a) Stack pointer
(b) Program counter
(c) CPU registers
(d) All of the above
(e) None of the above

Solution: All of the above □

1.4 Which of the following data structure is generally used by operating systems to store information associated to a process

(a) Process control block (PCB)
(b) Process identifier
(c) Process state
(d) All of the above
(e) None of the above

Solution: Process control block (PCB) □

1.5 An address generated by the CPU is commonly referred to as a/an ...

(a) Physical address
(b) Absolute address
(c) Logical address
(d) All of the above
(e) None of the above

Solution: Logical address □

1.6 Which of the following is an advantage of virtual memory?

(a) It allows the execution of processes that are not completely in memory
(b) It abstracts main memory into an extremely large logical memory
(c) It allows running programs that are larger than physical memory
(d) All of the above
(e) None of the above

Solution: All of the above □

1.7 Which threading model will allow all threads to run at the same time, as well as all CPU cores to be utilised at the same time?

(a) One-to-one
(b) One-to-many
(c) Many-to-one
(d) Many-to-many
(e) None of the above

Solution: One-to-one □

1.8 What is a kernel thread?

(a) A virtual CPU that user threads are mapped to
(b) What threads are called when used by the kernel
(c) Old-fashioned name for threads
(d) Used only for one-to-one threading model

Solution: A virtual CPU that user thereads are mapped to □

1.9 Why does it matters that your mobile phone is a Von Neumann machine?

(a) It does not matters at all

(b) It makes it possible to install software from app stores
(c) It means that it can perform any operation (Von Neumann-complete)
(d) Mobile phones are generally not Von Neumann-machines

Solution: It makes it possible to install sotftware from app stores □

1.10 What is external memory?

(a) Storage that is outside the computer cabinet
(b) Storage that is outside the CPU
(c) Storage that can not be directly adressed from the CPU
(d) Hot-pluggable storage (can be attached to a running computer)

Solution: Storage that can not be directly adressed from the CPU □

1.11 Which of the following is correct about a computer bus?

(a) It may connect more than two devices in a computer system
(b) It can not be serial (must have several parallel data lines)
(c) It must have high bandwith
(d) It is only used to connect the CPU to main memory
(e) All of the above

Solution: It may connect more than two devices in a computer system □

1.12 Which of the following is correct about CPU?

(a) A CPU die may only have one core
(b) A computer may only have one CPU die
(c) If a computer has more than one core, there may only be one CPU die in the computer
(d) A computer may have several CPU dies with several cores each

Solution: A computer may have several CPU dies with several cores each □

Problem 2 – Miscellaneous (16%)

2.1 Consider a computer system that uses 16-bit logical addresses. What is the size of the logical address space?

Solution: $2^{16} = 65536$ bytes □

2.2 Consider a computer system that supports a paging hardware with a translation look-aside buffer (TLB). Assume a process that has *some* of its requested pages in the physical memory. The page table of the process needs to be updated when there is a page fault. Assume further the process tries to access a page.

(a) How many memory accesses are needed to fetch the page if the page number is found in the TLB and there is no page fault?

Solution: One (for fetching the page) □

(b) How many memory accesses are needed to fetch the page if the page number is not found in the TLB and there is no page fault?

Solution: Two (one for looking up the page number from the page table, which is in the main memory, and one for fetching the page)
□

(c) How many accesses to the backing store if there is a page fault and the victim page is dirty.

Solution: Two (one for paging out the dirty page, and one for paging in the requested page) □

(d) When a process spends more time paging than executing, we say the process is ...

Solution: Thrashing □

2.3 Explain the differences and similarities between processes and threads. **Solution:** Both processes and threads may be utilised to perform some tasks in "parallell". For example if media needs to be downloaded at the same tame as it is played. However, there are also som differences. A process is a completelly independent "processing unit" - almost like a independent program. A thread on the other hand shares a number of resoures like the memory space and list of open files with it's parent process. Thus, a thread is much "cheaper" to create than a process. This means that a thread can easily excahnge data with i parent, while a process must utilise to process communication technique. It also means a bug in a thread can overwrite any part of the memory space of the parent, and thus cause the whole program to crash. A bug in a child process on the other hand can not write to the memory space of the parent. A crash in the child process does not need to casue a crash in the parent process. Depending on how the operating system implements threads, it may may be that only one thread of the program may run at any time. □

2.4 You have just started a new process using the `fork` system call. How do you determine if you are now executing in the parent or child process?

Solution: The fork system call will have zero as return value for the child, while for the parent process it will return the PID of the newly created child. □

2.5 What does it mean that a threading model is one-to-one? What are other possible threading models? Explain.

Solution: The one-to-one threading model means that each user thread will be mapped to a separate kernel thread. A kernel thread is essentially a virual core that can be used to execute the thread. The one-to-one threading model is rather costly, since a large number of kernel threads must be created. However, it is also very flexible as any threads can run at the same time. The many-to-one model mapps several user threads to one (same) kernel threads. Only one of these threads may run at any time. The many-to-one model maps a number of user threads to a pool of kernel threads. Similarly to the on-to-one model, it (may) means that any threads can run at the same time. However, this introduces a two-level schedueling system, that may make the schedueling less predictable. □

Problem 3 – Cache (15%)

3.1 Consider B_0, B_2, B_4, B_6 and B_8 are mapped to C_0, and B_1, B_3, B_5, B_7 and B_9 are mapped to C_1 where B_i refers to the i-th block in the main memory and C_j refers to the j-th cache line. Assume that B_1 and B_4 are in the cache. What are the number of cache misses if B_8, B_4, B_1, B_8, B_1 are accessed in the shown order? Justify your answer by identifying which accesses lead to cache misses.

Solution: B_8: miss; B_2: miss; B_1: hit; B_8: miss; B_1: hit

Hits: 2; misses: 3 □

3.2 Explain the purpose of the cache memory. What would the implications of a CPU not having cache memory be?

Solution: Although the system memory is very fast compared to the hard drive or even SSD, it is still very slow compared to accessing the CPU registers. Retrieveing data from the main memory is

thus an "expensive" operation. To overcome this bottleneck, the cache memory has been inserted between the CPU and the main memory. It has a number of "lines", each capable of storing a portion of the system memory. Each time some sytem memory is accessed, a copy of this data is written to the cache memory. For reading operations, it means a copy is written to the cache as well as to the CPU. For writing operations, the data is written to cahce rather than to system memory. The cache memory is of limited size, so only the most frequent or recent portions of the system memory is stopred in the cache. The cache memory is often on the same die as the CPU. There may be several layers of increasingly slower and larger cache memory. If there was no cache memory, data from the system memory could only be accessed at the speed of the memory bus, which would signigicantly slow down the computer operation. □

3.3 You want to buy a new computer. You are considering a model with several options for amounts of main memory and cache memory (but otherwise identical specifications). What are the implications of increasing the size of the cache memory compared to increasing the size of the main memory?

Solution: A computer may perform many different tasks. Some of these tasks will require more memory than others. If one is intending to run tasks which require much system memory, more system memory is required. Some other tasks may not not require a lot of system memory per se, but they may frequently access different portions of the system memory. Such tasks may benefit more from a larger cache memory, as there will be less cache misses, and the computer system will thus operate more efficiently. To decide whether more system memory or cache memory is beneficial for the computer, one needs to know how the tasks the computer will typically will run will utilise the memory. □

3.4 You are developing some software. One of your colleagues looks at your code, and claims that the way you have written the code will lead to a lot of cache misses. What could the reason for this be?

Solution: There can be several reasons for this. A cache miss happens when we try to access a portion of system memory which is not stored in the cache. To minimise the fruquency of cache misses, we should try to perform as many operations on the same data as possible, before switching to some other data. In other words, we should try to switch between different as rarely as possible. One possiblity is that we write the code in such way that it will first perform the same operation on a large number of data, then switch to some other operation. It would be better to instead perform as many operations on the same data before switching to some other data. □

3.5 Typically, a modern CPU has several layers of cache memory. Explain the reason for this - why not just a single layer?

Solution: In general, it is technically difficult or even impossible to make a memory both very fast and very large. The larger it is, the slower it will be. A cache memory that can operate more or less at the speed of the CPU can thus not be very large. At the same time, a cache memory that is large enough to fit a adequate part of the system moemory will not be fast enough, One overcomes this problem by dividing the cache memory into several layers - typically three on a modern CPU. These layers are increasingly slower and larger as they move away from the CPU. The operation principle is the same in all - the most frequently accessed data is cached in each level, and if not found, a search is performed on the next level of cache. □

Problem 4 – Assembly and Addressing Modes (11%)

4.1 Consider the assembly code fragment in Listing 1.

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

Listing 1

Assume all the instructions in Listing 1 have been executed, and the stack grows towards lower addresses.

(a) What is the value stored in register `ecx`?
Solution: 3 ☐

(b) What is the value stored in register `ebx`?
Solution: 2 ☐

(c) What is value stored in register `eax`?
Solution: 6 ☐

(d) What is the value at the address stored in `esp`?
Solution: 2 ☐

(e) What is the value at the address stored in `esp+4`?
Solution: 1 ☐

4.2 Identify the addressing mode for each of the operands used in the following instructions:

(a) `sub eax, 5`
Solution: `eax`: register; `5`: immediate ☐

(b) `mov ebx, [eax]`
Solution: `ebx`: register; `[eax]`: register indirect ☐

(c) `mov [esp + 4], NUM`
Solution: `[esp + 4]`: register indirect; `NUM`: direct ☐

Problem 5 – Process synchronisation (12%)

5.1 Typically, only one process can write to a *critical section* at the same time, while it is possible for several processes to read from a critical section at the same time. But what is the situation if one process wants to read while another wants to write to the same critical section at the same time? Explain.

Solution: If a process is writing to a critical section, it means that an other process can not read from it at the same time. It may be tempting to think that it will be possible for one process to read while another is writing, since there will not be a write race condition where to processes are competing to update the same data, there will still be a problem that we do not know if the reading process will have read the new or the old value. Presumably, this will not be indifferent. ☐

5.2 Three popular means to protect a critical section are *locks*, *semaphores* and *monitors*. Explain the differences and similarities between these methods.

Solution: While all of locks, semaphores and monitors can be used to protect a critcal section, they come with increasingly capabilitiesa and sophistication. The simplest mechanism is the lock. It is

a binary system. Either the lock is set or it is not. When it is not set, some process can enter the critical section, but it it is set, then the process has to wait for the lock to be opened. A semaphore can be thought of as a lock with more than just two states. It can count beyond "one", which may be useful if one for example has a buffer with a number of available slots, and one wants to assure that one does not go beyond tha boundaries of this buffer. A semaphore with jsut two states is essentially a lock. Thus, a semaphore can always be used in place of a lock, but the oposite is generally not true. A monitor is a more complex structure that for example can organise waiting queues. For locks and semaphores, busy waiting is typically utilised. With monitors, it is possible to suspend a waiting thread, and restart it when it can proceed to enter a critical section. Although this may be a more efficent approach, it is also more complicated and demanding to implement. □

5.3 One of your colleagues is developing some software with two processes exchanging data. This person has not used any process synchronisation mechanism, but is claiming that he/she never had any problems with the software dispite this. Explain to this person why this is not a good idea, and possible reasons why he/she might not have noticed any problems.

Solution: Depending a lot on the implementation details of the processes, it is possible that the probablity of a race condition in practices is very low (but not necessarily zero). For example if the data exchange is not very frequent, or it happens in certain patterns that also minimise the probability, Further, for some data it may be that the results of a race condition will not be very apparent in the output of the processing, even if it did occur. It may be that this person has so far been very lucky. However, it may be that on a different computer, or with different usage of the software, it may suddenly appear as "strange", unexpected behaviour. One should thus always use process synchronisation mthods, even if it "appears" to be working withouth. □

5.4 A software needs to protect a shared variable. Only one process can write to the variable at any time, while any number of processes can read from it at any time (of course, only when some process is not writing to it). Write pseudo-code for this. You can assume that basic functions needed for this protection is provided by system libraries.

Solution:

The writer process:

```
while (true) {
wait(rw mutex);
...
/* writing is performed */
...
signal(rw mutex);
}
```

The reader process:

```
while (true) {
wait(mutex);
read_count++;
if (read_count == 1)
wait(rw_mutex);
signal(mutex);
...
/* reading is performed */
...
wait(mutex);
```

```
read_count--;
if (read_count == 0)
signal(rw_mutex);
signal(mutex);
}
```

□

Problem 6 – CPU Scheduling (10%)

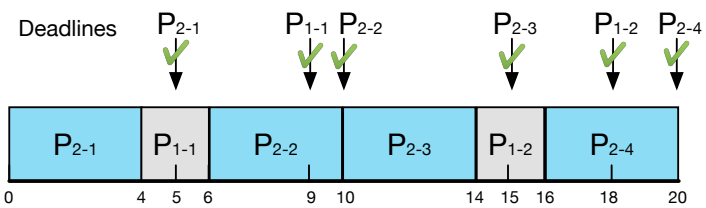
6.1 Consider the following table of processing time and period for two *periodic* processes P_1 and P_2:

Process	Processing time	Period
P_1	2	9
P_2	4	5

The completion deadlines of each process are the beginning of its next period. For example, the completion deadlines of P_1 are at time $9, 18, 27, \ldots$, while the completion deadlines of P_2 are at time $5, 10, 15, \ldots$.

Use *Earliest Deadline First* to schedule the two processes. At *time* $= 20$, can the two processes meet all their deadlines? Draw the Gantt chart of the execution to justify your answer.

Solution: Both P_1 and P_2 meet all their deadlines at time = 20. Note that at time=10, both P_{1-2} and P_{2-3} are available for scheduling, but P_{2-3} will be selected because it has an earlier deadline at time=15, while P_{1-2} has the deadline at time=18.



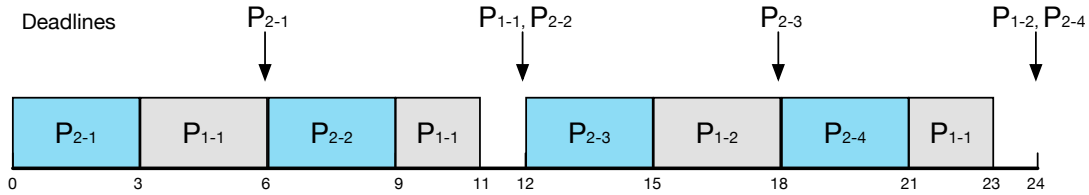
□

6.2 Consider the following table of processing time and period for two *periodic* processes P_1 and P_2:

Process	Processing time	Period
P_1	5	12
P_2	3	6

The completion deadlines of each process are the beginning of its next period. For example, the completion deadlines of P_1 are at time $12, 24, 36, \ldots$, while the completion deadlines of P_2 are at time $6, 12, 24, \ldots$. Use *Rate-Monotonic Scheduling* to schedule the two processes, where the priority is assigned based on the period of each process: ***the shorter the period, the higher the priority***. At *time* = *24*, can the two processes meet all their deadlines? Draw the Gantt Chart of the execution to justify your answer.

Solution: Both P_1 and P_2 meet their deadlines at time=24.



□

Problem 7 – Paging (12%)

7.1 Assume the page size is 2 KB[1]. Consider the following table:

Page number (i)	Frame number (j)	Frame starting address (k)
5	4	8192
6	12	24576
7	6	12288
8	5	10249

Each entry of the table represents a mapping from Page i to Frame j, where Frame j starts from address k in the physical memory. For example, the first entry refers to Page 5 is mapped to Frame 4 that starts from address 8192 in the physical memory.

(a) Write down, in decimal, the *page number*, the *offset* and the *physical address* for the logical address 14378.

Solution: page number: 7, offset: 42, physical address: 12330 □

(b) Write down, in decimal, the *logical address* for the physical address 9216.

Solution: logical address: 11264 □

7.2 Now assume the page size is 4 KB.

(a) If a process requests 20598 bytes from the operating systems, how many pages will the process be allocated? Is there any internal fragmentation?

Solution: $\frac{20598}{4096} = 5,029 \implies$ 6 pages will be allocated (3978 bytes of internal fragmentation) □

(b) Consider a computer system that uses 16-bit logical addresses. How many entries are there in a page table? What is the size of the page table?

Solution:

$\frac{2^{16}}{2^{12}} = 2^4 = 16$ entries;

size: 16×2 bytes = 32 bytes or 256 bits (each entry is 16 bits (2 bytes)) □

[1] 1 KB is 1024 bytes

Vedlegg – ASCII-tabell (fra www.asciitable.com)

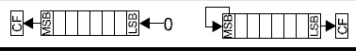
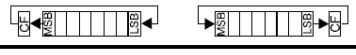
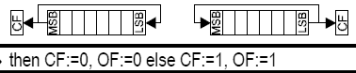
Dec	Hx	Oct	Char		Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL	(null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH	(start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX	(start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX	(end 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	&#127;	DEL

Source: www.LookupTables.com

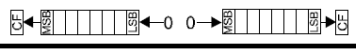
CodeTable 1/2

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*	Input	IN Dest, Port	AL/AX/EAX := byte/word/double of specified port									
OUT *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*				±	±	?	±	±
SHR	Shift logical right	SHR Op,Quantity		*i*				±	±	?	±	±

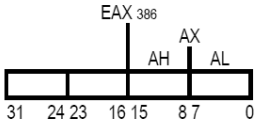
CodeTable 2/2

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					

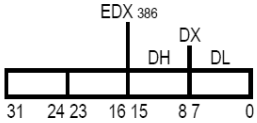
JUMPS (flags remain unchanged)							
Name	**Comment**	**Code**	**Operation**	**Name**	**Comment**	**Code**	**Operation**
CALL	Call subroutine	CALL Proc		RET	Return from subroutine	RET	
JMP	Jump	JMP Dest					
JE	Jump if Equal	JE Dest	(≡ JZ)	JNE	Jump if not Equal	JNE Dest	(≡ JNZ)
JZ	Jump if Zero	JZ Dest	(≡ 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)			
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	

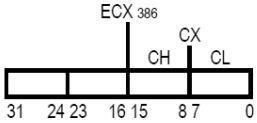
General Registers:



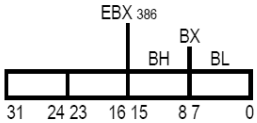
Accumulator



Data mul, div, IO



Count loop, shift



BaseX data ptr

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: - - - - 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

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	-	-	-	-