

Høgskulen på Vestlandet

Institutt for datateknologi, elektroteknologi og realfag

Eksamen i: DAT103 – Datamaskiner og operativsystem
Eksamensdato: 12 desember 2022
Eksamenstid: 9:00 – 13:00
Lovlege hjelpemiddel: Kalkulator
Språk: Norsk bokmål

Dette oppgavesettet har 7 sider, utenom vedlegget.
Du kan svare enten på engelsk eller norsk.

*Lykke til!
Dag og Violet*

Oppgave 1 – Flervalgsoppgave

(24%)

For hver av flervalgsoppgavene nedenfor, velge ikke mer enn et av alternativene.

1.1 Hvem av de følgende er et kontroll- og statusregister?

- (a) Minneadresseregister
- (b) Instruksjonsregister
- (c) Minnebufferregister
- (d) Alle de ovennevnte
- (e) Ingen av de ovennevnte

1.2 Hvem av de følgende utsagnene er **feil**?

- (a) Multiprogrammering er mulig i et enkelt-prosessorsystem
- (b) Multiprogrammering er mulig i ei multiprosessorsystem
- (c) Multiprosessering er mulig i ei enkelt-prosessorsystem
- (d) Multiprosessering er mulig i ei multiprosessorsystem
- (e) Alle de ovennevnte
- (f) Ingen av de ovennevnte

1.3 Hvem av de følgende tidsplanlegging-algoritmene er generelt optimal for minimerende gjennomsnittlig ventetid?

- (a) "Round robin"
- (b) Først kommet først håndtert
- (c) Korteste jobb først
- (d) Alle de ovennevnte
- (e) Ingen av de ovennevnte

- 1.4 Hvem av de følgende tidsplanlegging-agoritmene er *ikke-avbruddbare*?
- (a) "Round robin"
 - (b) Først kommet først håndtert
 - (c) Første frist først
 - (d) Korteste-tid-igjen-først
 - (e) Ingen av de ovennevnte
- 1.5 En *sidefeil* hender når ...
- (a) Ei TLB miss hender
 - (b) Et avbrudd hender
 - (c) Den forespurte siden er funnet i hovedminnet og er skitten
 - (d) Den forespurte siden er funnet i hovedminnet og er rein
 - (e) Ingen av de ovennevnte
- 1.6 Ta utgangspunkt i et multiprosessorsystem som bruker symmetrisk multiprosessering (SMP). Hvem av de følgende er riktig om multiprosessor tidsplanlegging for dette systemet?
- (a) Hver prosessor i dette systemet er selv-tidsplanleggande
 - (b) Hver prosessor i dette systemet kan ha sin egen private kø av klare prosesser, og har en tidsplanlegger til å undersøke klar-køen og velge en prosess til å utføre
 - (c) Alle prosessorene kan være i en felles klar-kø, og hver prosessor har en tidsplanlegger for å undersøke klar-køen og velge en prosess til å utføre
 - (d) Alle de ovennevnte
 - (e) Ingen av de ovennevnte
- 1.7 Ta utgangspunkt i et datasystem som støtter logisk "cache". Hvem av de følgende er **riktig**?
- (a) Den logiske "cache" er lokalisert mellom minne- håndteringenhet (MMU) og hovedminne
 - (b) Den logiske "cache" lagrer data ved å bruke hovedminne fysisk adresse
 - (c) Prosessoren har direkte tilgang til den logiske "cache"
 - (d) Alle de ovennevnte
 - (e) Ingen av de ovennevnte
- 1.8 Ta utgangspunkt i to prosesser, en for en nettleser, og en for en tekstbehandler, som konkurrerer for CPU tid. Hvilken tidsplanlegger vil typisk være ansvarlig for å tidsplanlegge de to prosessene til CPUen??
- (a) Langtids-tidsplanleggeren
 - (b) Mellomtids-tidsplanleggeren
 - (c) Korttids-tidsplanleggeren
 - (d) Avhenger av operativsystemet
- 1.9 Hva er den mest vanlige måten å bruke operativsystemfunksjoner fra et brukerprogram?
- (a) Direkte via systemkall
 - (b) Indirekte via systembibliotek
 - (c) Implementere det fra bunnen uten å involvere operativsystemet
 - (d) Brukerprogram trenger typisk ikke operativsystemfunksjoner

- 1.10 Du har utviklet noe programvare i C for Linux datamaskin baserte på x86. Hvilke andre plattformer vil det vere “enkelt” å porte denne til?
- (a) Enhver datamaskin baserte på x86
 - (b) Ethvert operativsystem baserte på POSIX
 - (c) Enhver datamaskin baserte på x86 som bruker ethvert operativsystem baserte på POSIX
 - (d) Enhver datamaskin som bruker ethvert operativsystem
- 1.11 Hva betyr “prgrammering i maskinvare”?
- (a) Gammelt ord for å laste programmet fra en fil
 - (b) Programmet er lagret på en harddisk
 - (c) Programmet er lagret på et hullkort
 - (d) Programmet er i form av maskinvare
- 1.12 Hva er den “moderne” måten å overføre data inni en datamaskin?
- (a) Parallell buss punkt-til-punkt med pakker
 - (b) Seriell punkt-til-punkt forbindelse med pakker
 - (c) Parallell buss punkt-til-punkt uten pakker
 - (d) Seriell punkt-til-punkt forbindelse uten pakker
 - (e) Ingen av de ovenfor

Oppgave 2 – Diverse

(14%)

- 2.1 Du utvikler en programvare som trenger å komprimere mye data i bakgrunnen mens brukeren samhandler med et GUI. Dette kan gjøres ved å starte enten en ny tråd eller en ny prosess for å håndtere denne oppgaven. Hva er forskjellene på å la enten en tråd eller en prosess håndtere dette?
- 2.2 I problemet ovenfor virkar det som at koden for å komprimere data har et problem som gjør at den til tider krasjer. Vil dette påvirke avgjørelsen din om å bruke en tråd eller en prosess for å håndtere denne oppgaven? Forklar svaret ditt.
- 2.3 Anta at 10 multitrådede program kjører samtidig på et datasystem med 16 kjerner. Anta videre at de fleitrådede programmene er skrevet slik at de bruker *mange-til-en* trådmodell, og totalt har 100 brukertråder. Hva er maksimum antall kjerner som vil bli brukt til å kjøre de 10 programmene?
- 2.4 Hvem av de følgende kan være del av en CPU?
- (a) Programteller
 - (b) I/O kontroller
 - (c) Direkte minnetilgang (DMA)
 - (d) L1 “cache”
 - (e) Haug
 - (f) Stabelpeker

- 2.5 La ℓ_1 , ℓ_2 og ℓ_3 være låser, og x en delt int variabel initialisert til 10. Anta tre prosesser P_1 , P_2 og P_3 som samtidig kjører de følgende kodedelene:

```
1 // Kjørt av P1 //
2 acquire( $\ell_1$ );
3 x++;
4 release( $\ell_1$ );
```

```
5 // Kjørt av P2 //
6 acquire( $\ell_2$ );
7 x++;
8 release( $\ell_2$ );
```

```
9 // Kjørt av P3 //
10 acquire( $\ell_3$ );
11 x++;
12 release( $\ell_3$ );
```

List alle mulige verdier som x kunne ha når alle tre prosessene har avsluttet.

Oppgave 3 – Sideinndeling

(11%)

- 3.1 Ta utgangspunkt i følgende tabell:

| Sidetall (i) | Rammenummer (j) | Ramme sin startadresse (k) |
|-----------------|--------------------|-------------------------------|
| 0 | 12 | 49152 |
| 1 | 3 | 12288 |
| 2 | 7 | 28672 |
| 3 | 10 | 40960 |
| 4 | 2 | 8192 |
| 5 | 1 | 4096 |
| 6 | 0 | 0 |
| 7 | 9 | 36864 |

Hver linje i tabellen 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 referer til side 0 er tilordnet til ramme 12 som starter fra adresse 49152 i det fysiske minnet.

Anta at sidestørrelse er 4 KB¹. Skriv ned, i desimal, *sidenummeret*, *“offset”* og den *fysiske adressen* for de logiske adressene:

- (a) 20515
- (b) 3812

- 3.2 Ta utgangspunkt i et datasystem med et logisk adresserom på 65536 sider, som er tilordnet et fysisk minne på 2048 sider. Anta at sidestørrelsen er 2 KB.

- (a) Hva er størrelsen for en ramme?
- (b) Hvor mange biter må en logisk adresse minst ha?
- (c) Anta videre at hver tabellinje er 32 biter lang. Hva er maksimum størrelse for det fysiske minnet systemet kan adressere?

¹1 KB er 1024 bytes

Oppgave 4 – Assembler og adresseringmoduser**(8%)**

4.1 Ta utgangspunkt i kodefragmentet i assembler i listing 1.

```

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

```

Listing 1

Anta at alle instruksjonene i listing 1 har blitt utført, og stabelen vokser mot lavere adresser.

- (a) Hva er verdien ved adressen lagret i esp?
 - (b) Hva er verdien ved adressen lagret i esp+8?
 - (c) Har register eax nå en 32-biter esp+12. Om ikke, hvilken verdi har den nå?
 - (d) Har register ebx nå en 32-biter esp+8? Om ikke, hvilken verdi har den nå?
 - (e) Har register ecx nå en 32-biter esp? Om ikke, hvilken verdi har den nå?
- 4.2 Anta at hver instruksjon nedenfor har allerede blitt hentet fra hovedminnet og plassert i et av registerene. Hva er det *totale antall* minnetilganger som er nødvendig for å hente operandene for hver av instuksjonene?
- (a) add eax, ebx
 - (b) mov ebx, [esp]
 - (c) mov [esp + 12], eax

Oppgave 5 – CPU tidsplanlegging**(11%)**

5.1 Ta utgangspunkt i den følgende tabellen for “burst”-tid og ankomsttid for fire prosesser P_1 , P_2 , P_3 og P_4 :

| Prosess | “Burst”-tid | Ankomsttid |
|---------|-------------|------------|
| P_1 | 5 | 3 |
| P_2 | 8 | 0 |
| P_3 | 4 | 5 |
| P_4 | 3 | 2 |

Bruk gantt diagram for å forklare svarene dine for det følgende:

- (a) Anta at *korteste-tid-igjen-først* tidsplanlegging-algoritme blir brukt, og at tidsplanlegging blir utført bare ved ankomst eller fullførelse av prosesser. Hva blir rekkefølgen for fullførelse av de fire prosessene?
- (b) Anta at “round robin” tidsplanlegging-algoritme med tidskvantum 4 blir brukt. Hva blir rekkefølgen for fullførelse av de fire prosessene?
- (c) Hvilken tidsplanlegging-algoritme gir en lavere gjennomsnittlig ventetid av de fire prosessene, korteste-tid-igjen-først eller “round robin”? Rettferdiggjør svaret ditt ved å vise utregningene for den gjennomsnittlige ventetiden for hver tidsplanlegging-algoritme.

Oppgave 6 – Minnetilgangtid**(8%)**

- 6.1 Ta utgangspunkt i et datasystem som ikke støtter sideinndeling, og anta at prosessoren i systemet trenger 5 ns for tilgang til “cache”- minne og 50 ns for tilgang til hovedminnet. Anta at treff-ratioen for “cache” er 95%, og at hvert “cache”-oppslag tar ingen tid. hva er den gjennomsnittlige minnetilgangtid for prosessoren?
- 6.2 Nå ta utgangspunkt i et datasystem som støtter sideinndeling og har en sideinndelingmaskinvare med oversettelse “look-aside” buffer (TLB). Anta at sidetabellen og *alle sidene* en prosess trenger er i det fysiske minnet. Anta at prosentandelen av ganger som sidennummeret av interesse er funnet i TLB² er 90%, hvert TLB oppslag tar ingen tid, og minnetilgangtid for hver tilgang er 100 ns.
- (a) Hva er effektiv tilgangtid (EAT)?
 - (b) Nå anta at *to-nivå sideinndeling*-algoritme blir brukt for å strukturere sidetabellen, og det er en TLB miss når prosessen spør etter en side. Hvor mange minnetilganger er nødvendig for å hente denne siden?

Oppgave 7 – Cache**(12%)**

- 7.1 Ta utgangspunkt i at B_0, B_2, B_4, B_6 and B_8 er tilordna til C_0 , og B_1, B_3, B_5, B_7 and 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_3 og B_8 er i “cache”. Hva er antall “cache” miss om B_8, B_2, B_8, B_9, B_8 har tilgang i den viste rekkefølgen? Rettferdiggjør svaret ditt ved å vise hvilke tilganger som førte til “cache” miss.
- 7.2 Som programmerer kan du utnytte CPU-register ved å referere til deres navn, og du kan utnytte hovedminnet ved å referere til dres adresser. Men hvordan utnytter du “cache”? Grunngi svaret ditt.
- 7.3 En multitrådet programvare blir testet på to datmaskiner med liknende spesifikasjoner (e.g. frekvens, “cache”-størrelse, etc.), bortsett fra at en har en fysisk CPU med to kjerner, mens den andre har to fysiske CPUer med en kjerne hver. Programvaren kjører mye raskere på datamaskinen med en CPU med to kjerner. Det blir nevnt at forskjellen kan ha å gjøre med “cache”. Gi en forklaring på dette. Er det sannsynlig at trådene utveksler mye data eller ikke? Grunngi svaret ditt.
- 7.4 Noen sier at “cache” størrelsen ikke har noe å si dersom datamaskinen har mye hovedminne (i.e. “cache” størrelse har bare noe å si for en datamaskin med lite hovedminne). Er du enig eller ikke? Grunngi svaret ditt.

²Det blir også kalt TLB treff-ratio.

Oppgave 8 – Prosesssynkronisering

(12%)

- 8.1 Noen sier at de ikke forstår hvorfor prosesssynkronisering kan være et problem i det hele tatt. De sier: dersom en prosess har skrevet data til en variabel eller buffer i hovedminnet, hvordan er det mulig at en annen prosess ikke vet om dette? Gi en forklaring hvordan dette kan skje (på CPU-register-nivå). Hva er termen som blir brukt for dette?
- 8.2 Forklar de Spisende filosofers problem. I hvilket tilfelle vil det bli en vranglås? Foreslå en endring slik at vranglås ikke vil skje. Forklar svaret ditt.
- 8.3 Bruk koden nedenfor for semaforer for å implementere din løsning på de Spisende filosofers problem.

```
1 wait(int *semaphore){
2     while(*semaphore <= 0) {} ; // opptatt venting //
3     *semaphore --;
4 }
5
6 signal(int *semaphore){
7     *semaphore++;
8 }
```

- 8.4 Ved sidan av semaforer, er låsar en annen populær konstruksjon brukt for prosesssynkronisering. Samnlign semaforer og låser. Hva er forskjellene? Gi eksempel på situasjoner hvor den ene vil være bedre enn den andre.

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

| 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

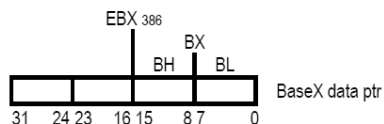
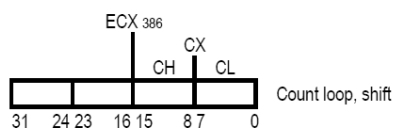
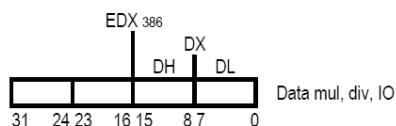
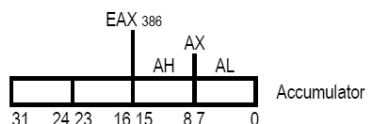
| LOGIC | | | | Flags | | | | | | | | |
|-------|---------------------------|-----------------|--|----------|---|---|---|---|---|---|---|---|
| Name | Comment | Code | Operation | O | D | I | T | S | Z | A | P | C |
| NEG | Negate (two-complement) | NEG Op | Op:=0-Op if Op=0 then CF:=0 else CF:=1 | ± | | | | ± | ± | ± | ± | ± |
| 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 (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 | 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) |
| 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 | - | - | - | - |