Programmering i maskinkode på *AMIGA*

A.Forness & N.A.Holten

Hæfte 8

Indhold

DATASKOLEN
Postboks 62
Nordengen 18
2980 Kokkedal

Telefon 49 18 00 77

Postgiro 7 24 23 44

AUDIO

I dette brev vil vi gennemgå AMIGA'ens lydmuligheder.

Hvad er lyd egentlig? Lyd er bølger, som forplanter sig i luften, akkurat som bølger i vand. Lydbølgerne har forskellige former, der afhænger af lydens karakter. En kraftig lyd har en høj bølgeform (høj AMPLITUDE). En næsten uhørlig lyd har en lang, lav bølgeform (lav AMPLITUDE). Derimellem findes et uendeligt antal variationer.

Et eksempel på en bølgeform kan du se i FIGUR 1 bagest i dette brev. I figuren ser du en helt almindelig bølgeform - en såkaldt "sinus"-bølge. Bølger af denne jævne type høres som en jævn (stabil) tone. Andre bølgeformer vil resultere i lyde af andre typer.

Nå, men så langt er det enkelt nok, men....

...hvordan laver en datamaskine lyd? Det gør den på (principielt) to måder. Den første metode er, at den bruger en såkaldt bølgeform-generator. Denne generator kan producere nogle forskellige bølgeformer. Commodore 64 benytter en sådan generator, hvor man kan vælge imellem: "sinus"-bølge, "trekant"-bølge, "savtands"-bølge og "støj"-bølge. Med disse grundformer kan man så få ganske komplicerede lyde (toner) frem. Man kan efterligne forskellige instrumenter som guitar, orgel, strygere (f.eks. violin), trompet etc. Jo mere avanceret bølge-generatoren er, jo bedre er den i stand til at efterligne forskellige instrumenter.

Den anden måde at producere lyd på er SAMPLE-metoden. Det er den metode AMIGA anvender.

SAMPLE, hvad er så det? En SAMPLE er også en bølgeform, men den har ikke nogen fast defineret form sådan som "sinus bølgen har. Hvis vi skal efterligne f.eks. et klaver vil det blive en meget ringe efterligning, hvis vi bruger en ren "sinus"-bølge. Pianostrengen producerer en mængde forskellige sinus-bølger på en gang. Disse bølger ligger over og under hinanden på en måde der er speciel for dette instrument.

Ethvert instrument har sin helt specielle sammensætning af mere eller mindre rene sinus-bølger. En trompet sætter også en mængde sinus-bølger sammen for at frembringe lyd. Men den sætter dem sammen på en helt anden måde. Det er derfor det menneskelige øre kan høre forskel.

Da pianolyden ikke kun består af en enkelt sinus-bølge (en helt perfekt og hel jævn tone), men har et lidt specielt "anslag" skal vi benytte SAMPLE-metoden for at få pianolyden god.

Når man bruger SAMPLE-metoden, skal man "indspille" lyden i AMIGAen. Det gøres ved hjælp af en en såkaldt SAMPLER. En SAMPLER kan også kaldes en A/D-CONVERTER. En "A/D-CONVERTER"

hedder på "dansk" en "analog/digital-omformer" fordi den laver et ANALOGT signal om til et DIGITALT signal.

Lad os starte med at forklare hvad en DIGITAL værdi er. En DIGITAL værdi kan kun være 0 eller 1 (akkurat som i datamaskiner). En ANALOG værdi (ofte en spænding) kan derimod indeholde uendelig mange værdier mellem 0 og 1 - en spænding kan jo være af hvilken som helst styrke mellem 0 og 1 (hvis dette er ydergrænserne), f.eks. 0.0001, 0.2, 0.314.....osv.

Alle SAMPLERE til AMIGA er såkaldte 8-BITs SAMPLERE. Det vil sige, at det ANALOGE signal ("bølgeformen") som kommer ind i SAMPLEREN bliver lavet om til en 8-BITs værdi før den kommer ud igen og derefter sendes ind i datamaskinen. Som bekendt er en gruppe på 8 BITs det samme som en BYTE - som kan indeholde værdier fra 0 til 255 (eller SIGNERET fra -128 til 127).

Studer FIGUR 2 bagest i brevet. Det er et eksempel på, hvordan en SINUS-bølge SAMPLES. Læg mærke til, at vi her bruger en SIGNERET værdi til at angive SAMPLE-dataene. De vertikale streger angiver, hvor det ANALOGE signal bliver lavet om til et DIGITALT signal. På den måde kan AMIGAen aflæse formen direkte fra det ANALOGE signal (bølgeform), og lagre det i hukommelsen (eller på disketten) som en DIGITAL værdi, dvs som 1 (en) eller 0 (nul).

For at få AMIGAen til at tilbagespille en SAMPLE (en SAMPLET lyd) benytter den sig af en indbygget D/A-CONVERTER (dansk: DIGITAL/ANALOG-former). FIGUR 3 viser, hvordan det ANALOGE signal vil se ud, når det afspilles på AMIGAen. Som du ser bliver signalet ikke så jævnt, som det var fra starten (FIGUR 2). Vi kan forbedre dette ved at aflæse det ANALOGE signal noget oftere (med A/D-CONVERTEREN). Det resulterer i, at de vertikale afmærkninger i figuren vil stå tættere sammen (v± får flere aflæsningspunkter). Dermed får vi en mere nøjagtig gengivelse af lyden/musikken som afspilles.

Der er dog den ulempe ved SAMPLING, at det bruger meget hukommelse. Hvis vi SAMPLER en lyd 10000 gange i sekundet - som er en ganske almindelig RATE (engelsk udtale: reit, og betyder hastighed) på AMIGA, vil en lyd på 5 sekunder "forbruge" 50000 BYTEs (eller ca. 49 kB). Vi kan nævne at en almindelig COMPACT DISK-spiller (som også benytter sig af SAMPLING-metoden bruger en 6 BITs D/A-CONVERTER med en SAMPLINGs-hastighed på 44.100 gange i sekundet. Hvis det var teknisk muligt på AMIGAen, ville 5 sekunder med lyd bruge 88200 BYTEs (ca. 86 kB).

Efter denne korte indføring i hvordan lyd kan laves med datamaskiner, og hvordan SAMPLING virker, går vi videre med temaet "Hvordan bruges SAMPLING i AMIGA?".

AUDIO II

I AMIGA findes der 4 lyd-kanaler. Det vil sige, at der kan spilles op til 4 forskellige lyde/eller SAMPLEs) samtidig. VOLUMEN (og afspilningshastigheden) kan justeres uafhængig for hver kanal.

Register-opsætningen for lyd-delen:

AUDxLEN registrene:

AUDIO-pointeren i AMIGA bruges til at pege på starten af SAMPLE-dataene i hukomme1sen.

BETEGNELSE	BITS	ADRESSE
AUD0LCH	16-31	$DFF0A0
AUD0LCL	0-15	$DFF0A2
AUD1LCH	16-31	$DFF0B0
AUD1LCL	0-15	$DFF0B2
AUD2LCH	16-31	AUDIO-pointeren

Læg mærke til at pointerne består af "to" registre (HIGH og LOW) på samme måde som f.eks. BITPLANE-pointerne.

AUD betyder AUDIO (lyd). EN betyder LENGTH (længde) og "x" angiver hvilken lydkanal (0,1,2 eller 3), der skal benyttes. Dette register benyttes til at fortælle AMIGA, hvor lang SAMPLEn er.

BETEGNELSE	ADRESSE
AUD0LEN	SDFF0A4
AUD1LEN	SDFF0B4
AUD2LEN	

AUDxLEN registrene

Registret AUDxLEN kan indeholde en 16-BITs værdi (0-65535). Det eneste du skal huske er at længden opgives i WORDS. Altså: Hvis du har en SAMPLE på 5000 BYTEs længde, skal du lægge 2500 ind i dette register (HUSK: 1 WORD = 2 BYTES).

Opsætningen for AUDxPER(PERIOD = periode/hastighed), "x" står for kanal 0,1,2 eller 3). Dette register benyttes til at give AMIGA information om SAMPLENs "afspilningshastighed .

BETEGNELSE	ADRESSE
AUD0PER	$DFF0A6
AUDIPER	$DFF0B6
AUD2PER	$DFF0C6

AUDxPER register

Det andet register AUDxPER indeholder hastigheden som SAMPLEn skal spilles med. Registret kan indeholde værdier fra 124 til 65535, hvor 124 er HØJESTE hastighed og 65535 er LAVESTE hastighed. Hvis du sætter dette register til en værdi mindre end 124, vil AMIGAen springe en del SAMPLE-data over, så lyden bliver forkert (selv om det ikke altid kan høres). Vi vil forklare mere om dette register senere i brevet når vi forklarer program-eksemplerne.

Opsætningen af AUDxVOL registrene (VOLUME = volumen/lydstyrke), "x" angiver kanal 0,1,2 eller 3. Disse registre benyttes til at bestemme lydstyrken på lydkanalerne.

BETEGNELSE	ADRESSE
AUD0VOL	$DFF0A8
AUD1VOL	$DFF0B8
AUD2VOL	$DFF0C8

AUDxVOL registrene

BIT 0-5 danner en BIT-gruppe som kan indeholde værdier fra 0 til 63. Volumen (lydstyrken) kan varieres "trinløst" fra 0 til 63 (64 forskellige trin) Så langt burde alt være godt. Hvis BIT 6 sættes til "1" ignoreres værdierne i BIT 0-5 af AMIGA og man får den højest mulige volumen (lydstyrke). Forestil dig BIT 0-6 som en BIT-gruppe og du kan variere lydstyrken fra 0 til 64 (65 forskellige trin). BIT 7-15 benyttes ikke.

I det følgende kapitel kaster vi os over program-eksemplerne MC0801 og MC0802 og forklarer disse mere indgående.

AUDIO III

Gennemgang af programeksempel MC0801:

Linie 1: Denne MOVE slukker AUDIO-DMA'en til lydkanal 0 (hvis den skulle være tændt). Se også opsætningen af DMACON i BREV III.

Linie 3: Henter adressen på "sample" ind i A1.

Linie 4: Lægger adressen på "sample" (A1) ind i $DFF0A0

Linie 5: Lægger værdien 48452 ind i $DFF0A4 (AUD0LEN). Altså: SAMPLEn vi skal spille er 48452 * 2 = 96904 BYTEs lang (ca. 95 kB).

Linie 6: Sætter spillehastigheden til 700 (AUD0PER).

Linie 7: Sætter volumen (styrken) til 0 (AUD0VOL).

Linie 9: Tænder AUDIO-DMA'en til lydkanal 0.

Programlinierne 11-22 øger gradvist volumenen til fuld styrke, og spillehastigheden øges også gradvist fra 700 til normal hastighed. Den normale spillehastighed er 180. Denne effekt gør, at det lyder som en pladespiller, som bruger lidt tid førend den kommer op på normal hastighed.

Programlinierne 28-37 virker modsat. Altså: Hastighed og volumen sænkes gradvist, så det lyder som om man pludselig river stikket ud mens grammofonen spiller.

Linie 11: Lægger 0 ind i D1. D1 bruges som tæller for volumenen.

Linie 12: Lægger 700 ind i D2. D2 bruges her for at tælle spillehastigheden gradvist ned til 180 (dvs. øge hastigheden).

Linie 13: Lægger 64 ind i D7 (dette register bruges som en LOOP-tæller). Altså: LOOPen udføres 65 gange (HUSK! 0 til 64).

Linie 16-17: Hopper til rutinen "wait". Dette gøres 2 gange. PÅ den måde får vi en pause på 1/25 sekund (2/50 sekund).

Linie 18: Lægger værdien i D1 ind i $DFF0A8 (AUD0VOL).

Linie 19: Lægger værdien i D2 ind i $DFF0A6 (AUD0PER).

Linie 20: Lægger 1 til værdien i registret D1. Linie 21: Trækker 8 fra værdien i D2.

Linie 22: Trækker 1 fra værdien i D0. Hvis D0 er større end -1, hop op igen til "up".

Linie 25-26: Disse to efterhånden velkendte linier venter til du har trykket på venstre musknap.

Linie 28: Værdien 64 lægges ind i D7 således at vi er klar til en ny LOOP.

Linie 31-32: Hopper til rutinen "wait" to gange så vi igen får en pause på 1/25 sekund.

Linie 33: Lægger værdien i D1 ind i $DFF0A8 (AUD0VOL).

Linie 34: Lægger værdien i D2 ind i $DFF0A6 (AUD0PER).

Linie 35: Trækker 1 fra værdien i D1 (volumen-tæller).

Linie 36: Lægger 8 til værdien i D2 (hastigheds-tæller)

Linie 37: Trækker 1 fra værdien i D0. Hvis D0 er større end -1, hop op igen til "down".

Linie 39: Slukker AUDIO-DMA'en til lydkanal 0.

Linie 41: Afslutter programmet.

Linie 43-55: Disse skulle være kendt stof for alle nu, men du undrer dig måske over, hvorfor vi venter både på linie 200 og på linie 201? Det er fordi rutinen skal udføres to gange i træk. Forestil dig at vi havde fjernet programlinierne 50-55. Første gennem kørsel ville gå godt. Men anden gang (lige efter) vil elektronkanonen holde op med at tegne linie 200, så rutinen vil blive afsluttet for tidligt.

Linie 60: Her har vi sat plads af til SAMPLE-dataene, som ligger på KURSUSDISKETTE 1.

Før programmet kan køres, må det først og fremmest assembles. Derefter læser du filen "SAMPLE" ind, som ligger i DIRECTORY "BREV 08" på kursusdisketten, og til sidst starter du programmet med "j". Det gøres sådan (vi viser hvordan det vil se ud på skærmen).:

```
SEKA>a
OPTIONS
No errors
SEKA>ri
FILENAME>brev08/sample
BEGIN>sample
END>
SEKA>j
```

Tips: Hvis du vil køre programmet flere gange uden at være nødt til at at assemble det først, skal du lægge en LABEL ind på den første linie i programmet. Det kan f.eks. se sådan ud:

```
1 start:
```

Derefter skriver du sådan hver gang du vil starte programmet forfra:

```
SEKA>jstart
```

("jstart" betyder altså JUMP (hop) til label "start" og kør programmet derfra.)

Vi håber denne forklaring hjælper dig til at forstå, hvad der foregår i maskinen, når en SAMPLE skal afspilles. Glem ikke at eksperimentere med de forskellige værdier i programmet for at få nye effekter frem. Det er meget lærerigt!

Og nu fortsætter vi med forklaringen til programeksempel MC0802:

Linie 1: Slukker AUDIO DMA'en til lydkanal 0.

Linie 3: Lægger adressen på "sample" ind i A1.

Linie 4: Lægger værdien i A1 ind i $DFF0A0 (pointeren til lydkanal 0).

Linie 5: Sætter længden på SAMPLEn til 8 WORDs (16 BYTEs).

Linie 6: Lægger 0 ind i $DFF0A6 (AUD0PER).

Linie 7: Lægger 0 ind i $DFF0A8 (AUD0VOL).

Linie 9: Tænder for AUDIO DMA'en til lydkanal 0.

Linie 11: Lægger adressen på "music" ind i A1.

Linie 13: Her starter rutinen, som spiller de forskellige toner.

Linie 14: Hopper til rutinen "wait". Denne rutine vil lave en pause på 5/50 sekund (en tiendedels sekund).

Linie 16: Lægger værdien, som ligger på adressen, som A1 peger på ind i D1; derefter lægges 2 til værdien i A1 (Husk ! MOVE.W.).

Linie 17: Lægger værdien i D1 ind i $DFF0A6 (AUDOPER).

Linie 18: Henter næste værdi i tabellen "music", og lægger værdien i D2.

Linie 19: Lægger værdien i D2 ind i $DFF0A8 (AUDOVOL).

Linie 21: Sammenligner D1 med 0.

Linie 22: Hvis D1 ikke er 0, hoppes der tilbage til "mainloop".

Linie 23 Sammenligner D2 med 0.

Linie 24: Hvis D2 ikke er 0, hoppes der op igen til "mainloop". Altså: Hvis både D1 og D2 er lig med 0, vil programmet fortsætte frem til linie 26.

Linie 26: Slukker AUDIO DMA'en til lydkanal 0.

Linie 27: Afslutter programmet.

Linie 29: Her begynder den rutine, som laver en pause på 5/50 sekund (1/10 sekund).

Linie 30: Lægger værdien 4 ind i D1. D1 bruges her som tæller (den "venter" 1/50 sekund 5 gange).

Linie 33-37: Venter til elektronkanonen er nået til skærmlinie 200.

Linie 40-44: Venter til elektronkanonen er nået til skærmlinie 201. Grunden til at man skal vente både på skærmlinie 200 og på 201 er forklaret i gennemgangen af programeksempel MC0801.

Linie 46: Trækker 1 fra værdien, som ligger i Dl. Checker om Dl er -1. Hvis ikke, hoppes der op igen til "wait2". Altså: Programlinierne 33-44 bliver udført 5 gange (derfor 5 * 1/50 sekund).

Linie 48: Hopper tilbage til programlinie 14, og fortsætter derfra.

Linie 51: Her ligger SAMPLEn, som spilles. Disse data svarer til en SINUS-bølge. Læg mærke til, at AUDIO DMA'en automatisk spiller SAMPLEn om og om igen sådan, at der kommer en uendelig lang sinus-bølge.

Linie 54-83: Her ligger dataene, som kontrollerer hvilke toner der skal spilles.

Den første værdi skal lægges ind i AUDOPER-registret.

Den anden værdi lægges ind i AUDOVOL-registret.

Den tredie værdi lægges ind i AUDOPER-registret, og så videre indtil alle data (toner) er afspillet.

Det der sker, når programmet kører, er at de to første værdier vil blive lagt ind i henholdsvis AUD0PER og AUD0VOL. Derefter vil programmet vente i 1/10 sekund, og de to næste værdier bliver lagt ind i AUD0PER og AUD0VOL. Således fortsætter det indtil der indlæses 0 i både AUDOPER og AUDOVOL, som tegn på at tabellen er slut, og programmet afsluttes.

Altså: Programlinierne 54-55 vil spille tonen "C" i 2 tiendedele sekund, derefter vil der komme et ophold (volumen sættes til 0) på et tiendedels sekund før næste tone afspilles. Bagest i brevet er en tabel, som indeholder værdierne for de forskellige toner i skalaen. Prøv at lægge dine egne toner ind.

I dette kapitel har vi gennemgået, hvordan man spiller en færdiglavet SAMPLE, og derefter hvordan man kan lave et enkelt lydprogram, som kan spille de forskellige toner. Nu er det din tur.

Husk ØVELSE GØR MESTER.

MASKINKODE VII

I denne maskinkode-del gennemgår vi instruktionerne MULU, MULS, DIVI og DIVS.

Lad os begynde med MULU, som står for: MULTIPLY UNSIGNED -eller multiplicer USIGNERET. MULU udfører en multiplikation med USIGNEREDE værdier.

```
MOVEQ #10,D1
MULU #5,D1
```

Første linie lægger værdien 10 ind i D1. Den anden linie multiplicerer D1 med 5, således at D1 vil indeholde 50. Læg mærke til at MULU multiplicerer to WORDs og giver et LONGWORD som resultat.

Den næste instruktion vi skal se på er MULS. MULS står for (som du sikkert allerede har gættet): MULTIPLY SIGNED, eller multiplicer SIGNERET. Forskellen på MULU og MULS er at MULS multiplicerer to SIGNEREDE WORDs og giver et SIGNERET LONGWORD som resultat.

Eksempel:

```
MOVEQ #10,D1
MULS #-5,D1
```

D1 vil nu indeholde -50 (FFFFFFCE).

Instruktionen DIVU som vi nu skal se på, står for: DIVISION UNSIGNED - eller divider USIGNERET. Denne instruktion tager et LONGWORD og deler det med/på et WORD, og giver et lidt specielt resultat, som vi vil se lidt nærmere på.

Men lad os først tage et eksempel:

```
MOVE.L #500,D1
DIVU #10,D1
```

og et til......

```
MOVEQ #10,D1
DIVU #3,D1
```

og endnu et......

```
MOVE.L #1001,D1
DIVU #2,D1
```

Det første eksempel vil give følgende værdi i D1: $00000032 (eller 50 DECIMALT)

Det andet eksempel tager tallet 10 og dividerer det med 3, som er lig med 3,33. Det ser lidt "rodet" ud, ikke sandt? I og med at det BINÆRE talsystem kun kan indeholde heltal, vil Dl se således ud: $000100003. BIT 0-15 ($0003) giver heltallet, og BIT 16-31 ($0001) giver antal "3die dele" i rest. (Havde vi divideret med 5, så ville BIT 16-31 indeholde antal femtedele, som var i rest). Altså bliver resultatet: 3 plus en trediedel, eller 3,33.

Det sidste eksempel vil heller ikke "gå op". D1 vil nu se således ud: $000101F4. Altså: 500 ($01F4 = 500 DECIMALT) plus 1 ($0001) halv, som kan skrives 500 1/2 eller 500,5.

Den sidste instruktion DIVS virker på samme måde som DIVU, men opererer med SIGNEREDE tal. Det går vi ud fra du selv kan slutte dig til på grundlag af det foregående.

Dette er for øvrigt instruktioner som, selv om de er meget effektive, ikke bliver benyttet så ofte. Det er mest når man skriver komplicerede programmer med en hel del matematiske beregninger, at de kommer til deres fulde ret.

Dette var så maskinkode-delen i dette brev. I næste brev ser vi på INTERRUPTS og de instruktioner, som bliver aktuelle i forbindelse hermed.

MIDI

I dette kapitel forklares lidt om hvad MIDI er. Forkortelsen MIDI står for "musical Instrument Digital Interface", (udtales "mjusikal instrument didjital interfeis" og betyder: Digitalt mellemled mellem musikinstrumenter).

Musikinstrumenter som benytter MIDI er mest udbredt i instrument-gruppen KEYBOARDS (elektroniske pianoer eller SYNTHESIZERS). Disse instrumenter kan kobles sammen via et MIDI-interface. De kan f.eks. kobles således, at når du spiller på det ene KEYBOARD, vil det andet spille nøjagtig det samme, men sikkert med en anden form for lyd. Man kan altså sige, at via MIDI kan du styre mange KEYBOARDS (eller SYNTHESIZERS) fra et KEYBOARD.

Hvis man går et skridt længere, kan man koble et KEYBOARD sammen med en datamaskine. Dette åbner for et hav af muligheder. Den mest almindelige brug af denne opkobling er at "indspille" f.eks. bassen i melodien først, for derefter at "afspille" bassen (som en PLAYBACK) mens næste stemme (f.eks. melodistemmen) spilles. På denne måde kan man opbygge en melodi, ja en hel symfoni, skridt for skridt.

Selve MIDI-overføringen er opdelt i kanaler (CHANNELS). Hvis man f.eks har to KEYBOARDS koblet sammen med en datamaskine, har man muligheden for at lade KEYBOARDene spille helt forskellige ting. Du lader bare KEYBOARDene "lytte" på forskellige kanaler. Et MIDI-interface har 16 sådanne kanaler.

Her kommer nogle tekniske forklaringer om MIDI. MIDI benytter sig af en SERIEL overføringsmetode. SERIELT betyder, at dataene som overføres, sendes eller modtages en BIT ad gangen. For at få en BYTE skal man altså sende 8 BITs efter hinanden. Overføringshastigheden for MIDI er 31250 BITS pr. sekund (betegnes BAUD), og udtales: BO). Altså, der kan overføres op til 31250/8 = 3906 BYTEs pr. sekund (i praksis vil der være et lille ophold mellem hver ottende BIT, som overføres, således at vi må regne med ca. 3000 BYTEs (ca. 3kB) i sekundet som maksimal overføringshastighed.

Denne overføringshastighed bliver mere end rigelig hvis man ser på, hvad som overføres via MIDI. Når man trykker en tast ned, overføres der kun 3 BYTEs. Den første BYTE fortæller hvilken kanal der sendes på, samt hvilken type information som overføres (i dette tilfælde "tast ned", eller "NOTE ON", som det hedder på MIDI-sprog). Den anden BYTE indeholder hvilken tast som blev trykket ned, mens den tredie BYTE indeholder, hvor hårdt den blev trykket (såkaldt VELOCITY).

Du har nu fået en enkel indføring i hvad MIDI er, og hvordan dette fungerer i al enkelthed. I BREV XII findes nogle enkle MIDI-rutiner til AMIGA. For at benytte disse rutiner skal du have et KEYBOARD (som har MIDI) - og et MIDI-INTERFACE).

LØSNINGER TIL OPGAVER I BREV VII

OPGAVE 0702:

BINÆRT	USIGNERET	SIGNERET
01101001	105	105
10111110	190	-66
11110001	241	-15
00101101	45	45
11010011	211	-45

OPGAVE 0703: Den mindste værdi en BIT-gruppe på 20 kan indeholde er -524388 (DECIMALT), og den højeste værdi er 524287 (DECIMALT).

OPGAVE 0704: Dette program kan f.eks. se således ud:

```
NOT.W         D1
ADD.W         #1,D1
RTS
```

OPGAVER TIL BREV VIII

OPGAVE 0801: Hvad indebærer AMPLITUDE i forbindelse med lyd?

OPGAVE 0802: Hvad er SAMPLING?

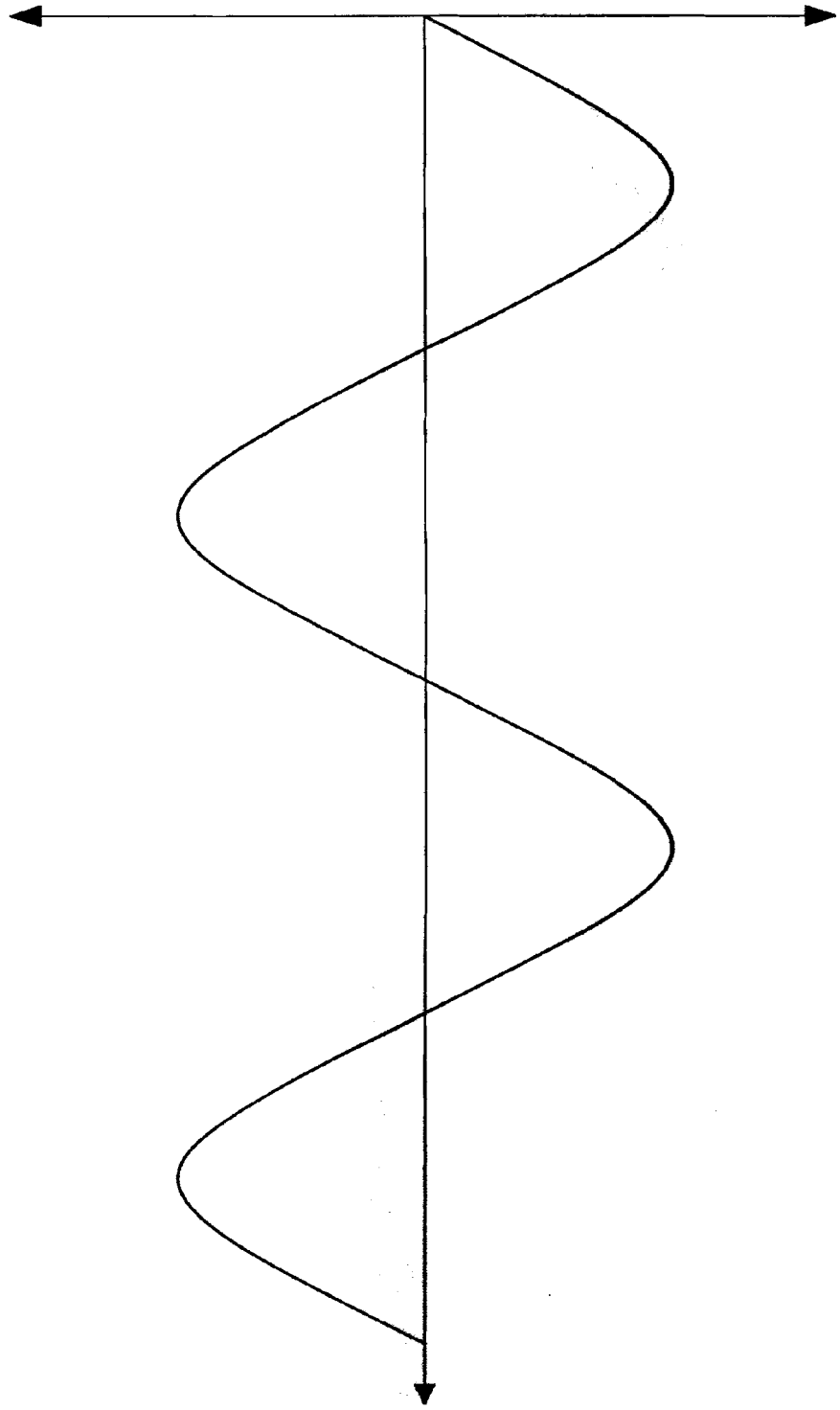
OPGAVE 0803: Hvad er en A/D-CONVERTER?

OPGAVE 0804: Hvad bliver resultatet i D1 efter at dette program er udført (prøv uden K-SEKA!):

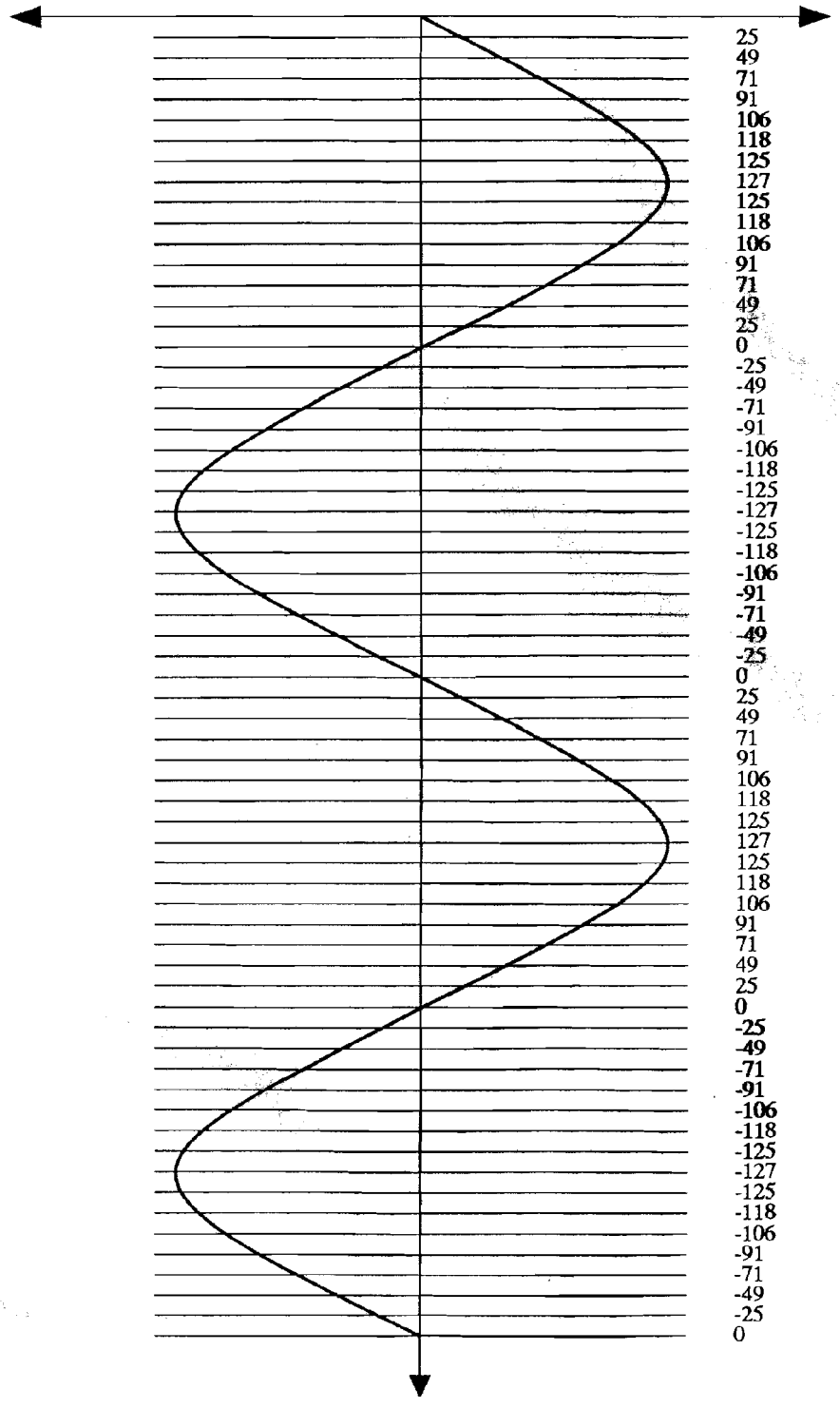
```
MOVE.W        #-75,D1
MULS          #-3,D1
RTS
```

TABEL OVER FORSKELLIGE TONERS SAMPLINGS-RATE

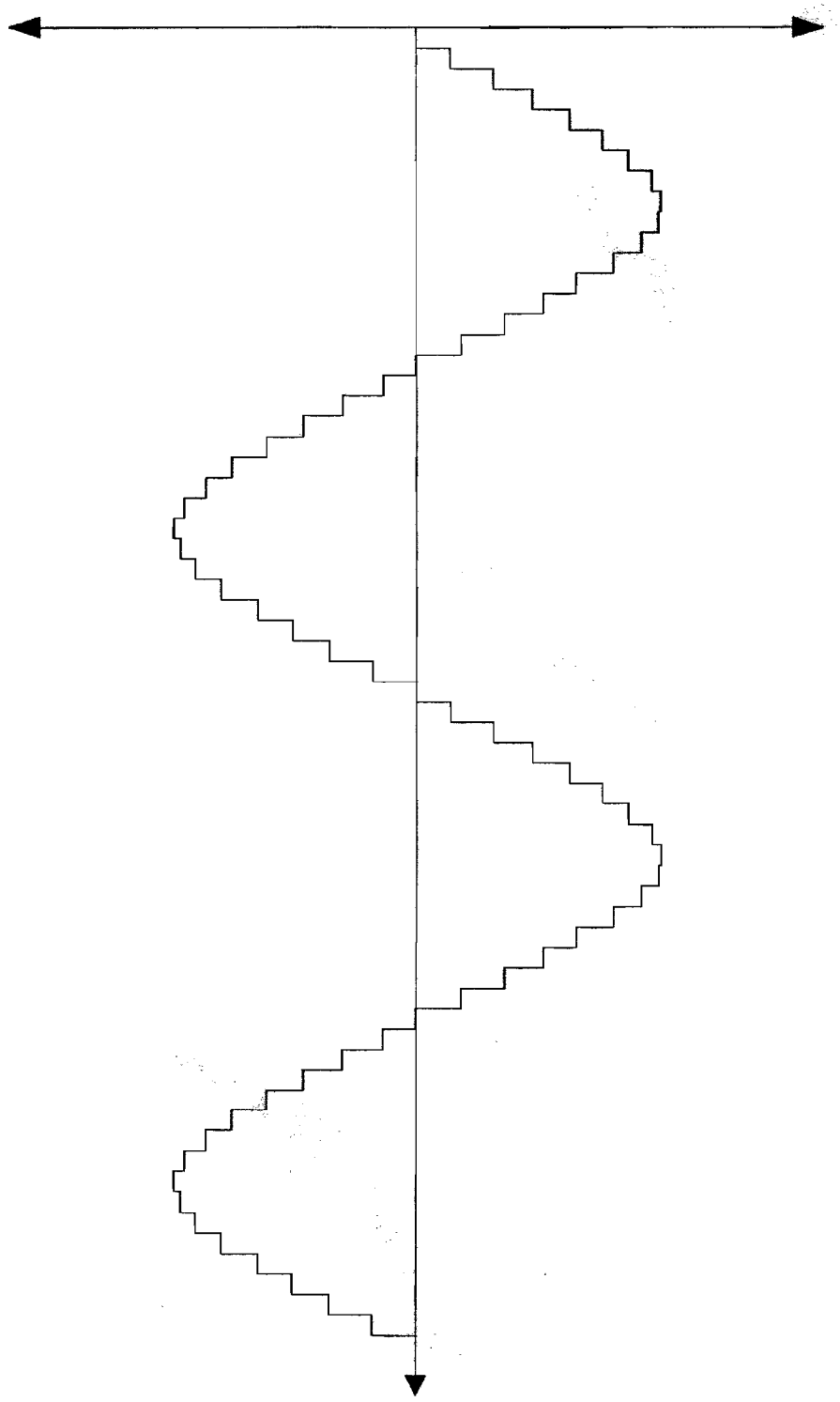
TONE	PERIODE **(RATE)**
A1	**508**
A1#	**480**
H1	**453**
C2	**428**
C2#	**404**
D2	**381**
D2#	**360**
E2	**339**
F2	**320**
F2#	**302**
G2	**285**
G2#	**269**
A2	**254**
A2#	**240**
H2	**226**
C3	**214**
C3#	**202**
D3	**190**
D3#	**180**
E3	**170**
F3	**160**
F3#	**151**
G3	**143**
G3#	**135**
A3	**127**



Figur 1.


25
49
71
91
106
118
125
127
125
118
106
91
71
49
25
0
-25
-49
-71
-91
-106
-118
-125
-127
-125
-118
-106
-91
-71
-49
-25
0
25
49
71
91
106
118
125
127
125
118
106
91
71
49
25
0
-25
-49
-71
-91
-106
-118
-125
-127
-125
-118
-106
-91
-71
-49
-25
0

Figur 2.



Figur 3.