Datorteknik TSIU02 Lab 2
Morsesändare — v0.7

Inledning

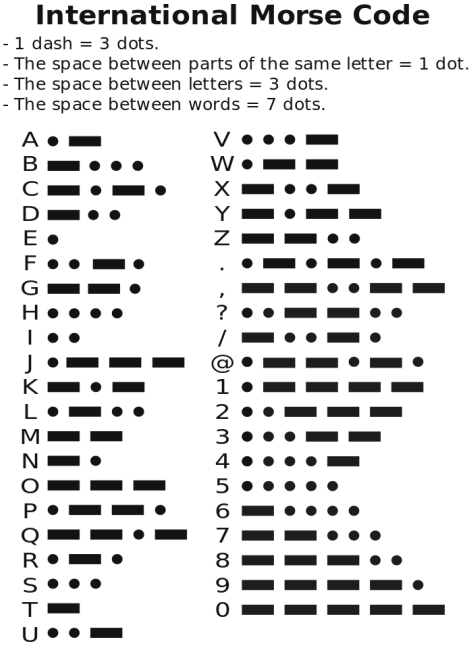
För att skriva program i något programspråk förenklar det att ha ett strukturerat angreppssätt. I assembler får man strukturen genom omsorgsfull användning av subrutiner.

Som exempel på en mer avancerad programmeringsuppgift som dessutom utnyttjar fördelarna med strukturerad programmering skall du i denna laboration tillverka en morsesändare.

Morsealfabetet uppfanns på 1850-talet av Samuel Morse som ett sätt att överföra information över telefontråd[1]. Den internationella varianten av alfabetet bestämdes något senare och innehåller vad informationsteoretikerna idag kallar huffmankodning.

I detta laboration-PM finns

- en beskrivning av hur morsetecknet kodas i binärkod,
- hur denna kodning skall tolkas till ljud, samt
- JSP-notationen för det slutliga programmets struktur.



Laborationsuppgifter

Laborationen består av tre uppgifter där den tredje är en extrauppgift. När **Uppgift 1** är klar har du en apparat som kan sända morsekod till en högtalare. I **Uppgift 2** och **Uppgift 3** genomför du sedan några modifikationer med JSP-notation som underlag.

Uppgift 1 Skriv ett program som, i morsekod, sänder ut den text som återfinns under labeln `MESSAGE`. Ljudet skall höras i en högtalare.

Ditt program skall ansluta till den övergripande JSP-strukturen i detta labhäfte. Du får själv komplettera med saknade subrutiner för tabelluppslagning och ljudgenerering. Programmet måste vara helt förberett innan laborationen. Fördröjningsrutinen från förra laborationen är lämplig att användas för att skicka en ton till högtalaren.

Tidsenheten för sändningshastighet och tonfrekvens skall definieras på **ett** ställe i programmets början (med ”`.equ`”).

Rutinerna för `BEEP` och `NOBEEP` skall vara i huvudsak samma kodrader. Längden (N, 2N, . . .) skall anges som argument till dem.

[1]Mikrofonen var ännu inte uppfunnen. . .

Uppgift 2 Hur tar ditt program hand om ett mellanslag (ASCII `$20`) i textsträngen? Studera strukturdiagrammet och implementera en ändring så att mellanslag utförs korrekt, dvs med totalt 7 tidsenheters tystnad mellan orden.

Uppgift 3 (Extra). Om det inte finns binärkod för ett önskat tecken skall programmet sända ett mellanslag i stället. Ett strukturdiagram för denna modifiering är given. Finns det någon annan (bättre?) modifiering för detta?

Programmets modulära struktur underlättar denna typ av ändringar. Ändringen skall kunna utföras som en modifikation utan större ingrepp i det befintliga programflödet. Skriv kompletterande assemblerkod för din lösning och provkör.

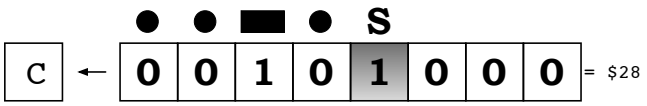
Morseteckenkodning

Morsetecknen består av korta och långa teckendelar i form av tonsignaler i olika kombinationer. Varje bokstav och siffra motsvaras av ett morsetecken som byggs upp av korta och långa tonsignaler.

En kort signal (”prick”/”dit”) är 1 tidsenhet lång, och en lång signal (”streck”/”dah”) är 3 tidsenheter. Mellan teckendelarna skall det vara 1 tidsenhets tystnad, mellan tecknen 3 tidsenheters tystnad och mellan ord 7 tidsenheters tystnad.

Tidsenheten bör vara ganska lång med datormått mätt, omkring 20 ms, för att sändningen skall kunna avlyssnas av människor.[2]

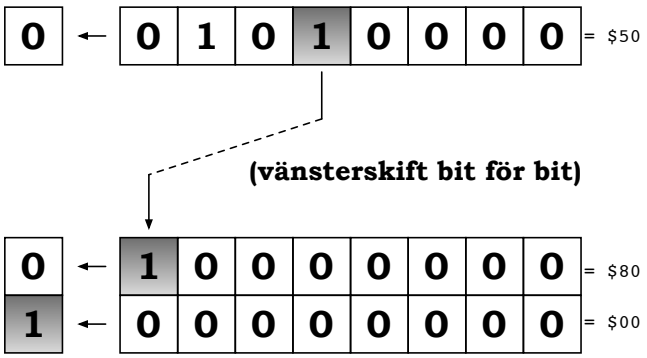
Morsetecknen är olika långa varför en speciell bytevis *binärkodning* används för dem i laborationen. I denna kodning är ettan längst till höger i byten en slutmarkering, `S`, som inte ingår i själva morsekoden. Till exempel blir binärkoden för bokstaven ”F” ($\cdot\cdot-\cdot$), med slutmarkering `S`:



Med denna kodning kan man skifta bitarna ett steg i taget åt vänster och låta innehållet i `C`-flaggan bestämma om man skall sända ut kort eller lång ton.

När den sista ettan, `S`, skiftats ut i `C`-flaggan är byten i övrigt noll, och man kan på så sätt detektera när hela tecknet är sänt.

Nedan visas detta, efter ett inledande vänsterskift, bit för bit på bokstaven ”F”:



Hårdvara

Laborationen använder hårdvaran:

- Högtalarmodul

Innan laborationen måste du skriva och simulera kod för:

- Hur man genererar en ton med enbart digital utsignal
- Hur data lagras i programminnet
- Hur data nås om det ligger i programminnet

Texten `MESSAGE`

Texten som skall sändas, `MESSAGE`, skriver du själv in i mikrokontrollerns programminne, FLASH.

För uppgift 1 kan texten vara:

```
MESSAGE:
    .db "DATORTEKNIK", $00
```

För uppgift 2–3 måste texten dessutom innehålla mellanslag och/eller okända tecken som till exempel ”`HEJ HOPP`” eller ”`HEJ HÖPP`”.

Strängen anges alltid i versaler[3] och avslutas med ett `NUL`-tecken, en byte som är binärt noll.

Binärkodning i programminne

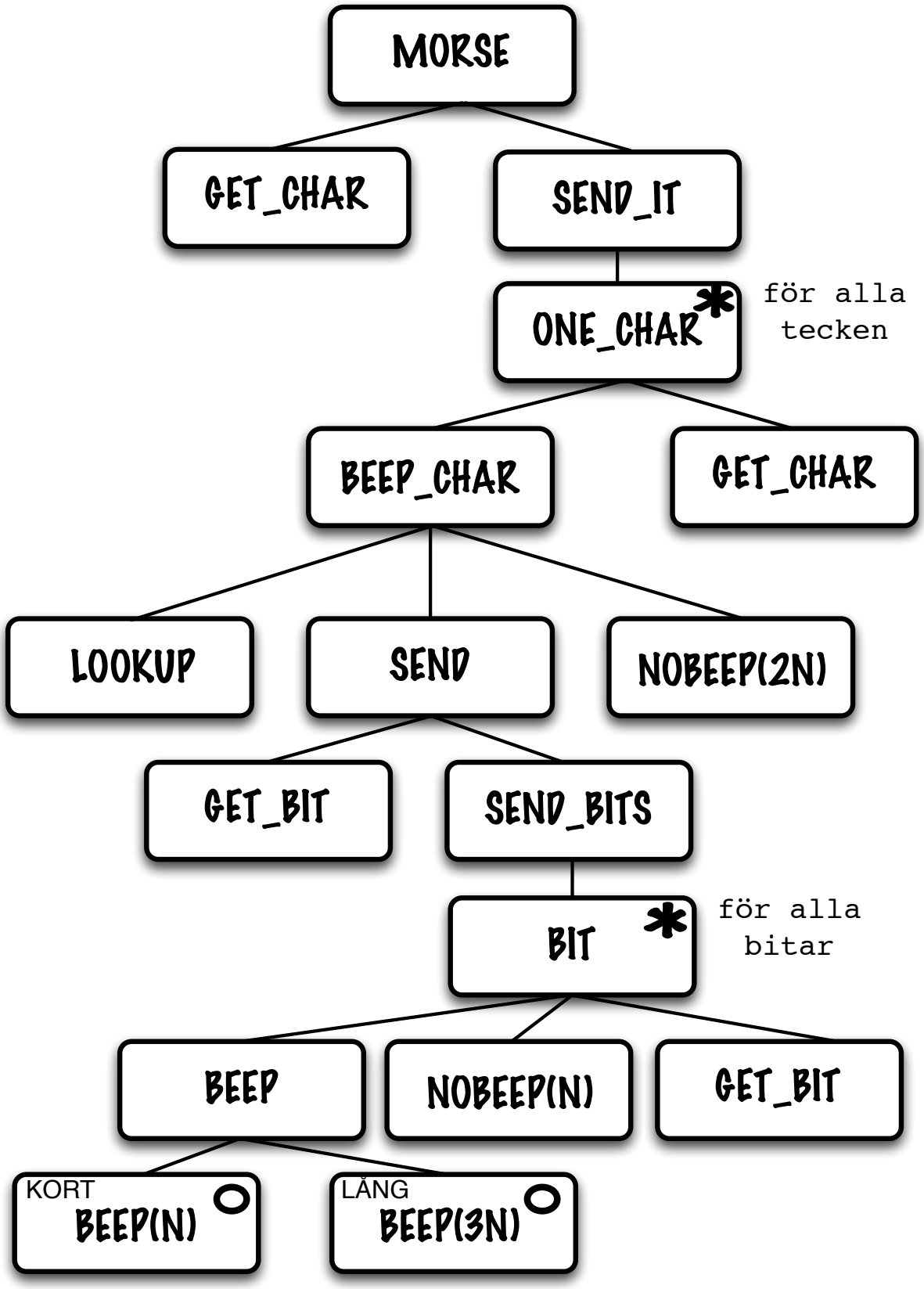
Översättningen mellan ASCII och den nödvändiga binärkodningen återfinns i slutet av detta häfte. Tabellen skall ligga i mikrokontrollerns programminne under labeln `BTAB` (för `BinärTABell`) och det är din uppgift att lägga den där.

Med kännedom om adressen för tabellstarten `BTAB`, kan programmet sedan nå rätt binärkod med indexerad adressering.

[2] En överföringshastighet på 60–150 tecken per minut är rimlig.
[3] Hur kan du tillåta även gemener i texten? Tips: Studera ASCII-tabellen.

Programstruktur JSP

Använd strukturdiagrammet när du skriver assemblerkoden. Som ledtråd till lösning finns pseudokod på nästa sida. Komplettera med de rutiner och variabler som behövs för ett komplett program. Övertyga dig om att strukturdiagrammet faktiskt löser uppgiften.



Programstruktur Pseudokod

Pseudokoden nedan är skriven utgående från strukturdiagrammet. Det kan vara onödigt att använda subrutiner för varje enskild ruta i diagrammet och pseudokoden har på detta sätt slagits ihop till två större kodavsnitt.

Notera att detta *inte* betyder att man bryter mot det strukturerade programflödet.

```
MORSE{
      CALL GET_CHAR;
      while( char != 0){        // Tills NUL
         CALL LOOKUP;
         CALL SEND;
         CALL NOBEEP(2N);       // 2N tystnad
         CALL GET_CHAR;
      }
}

SEND{
      CALL GET_BIT;             // Hämta nästa bit
      while ( Z != 0){          // tills hela sänt
           if bit==0
                CALL BEEP(N);  // 1N ljud, dit
           else
                CALL BEEP(3N); // 3N ljud, dah
           CALL NOBEEP(N);     // 1N tystnad
           CALL GET_BIT;
      }
}
```

Med bra namn på rutiner och variabler behöver man inte ha så många kommentarer. Om de valda namnen ändå skulle vara otydliga kommer här en kort beskrivning av namnen i strukturdiagrammet och pseudokoden.

Rutinnamn	Beskrivning
`MORSE`	Huvudprogrammet som sänder hela strängen.
`GET_CHAR`	Hämtar nästa ASCII-tecken ur strängen.
`ONE_CHAR`	Ombesörjer sändandet av ett ASCII-tecken.
`BEEP_CHAR`	Sänder ett morsetecken.
`LOOKUP`	Översätter ASCII-tecken till binärkod.
`SEND`	Skickar iväg ett tecken.
`GET_BIT`	Hämtar nästa bit för sändning.
`SEND_BITS`	Skickar iväg alla bitar för ett tecken.
`BIT`	Sänder kort eller lång bit med efterföljande tystnad.
`BEEP(N)`	Piper i högtalaren tiden N.
`NOBEEP(N)`	Piper *inte* i högtalaren tiden N.

Du får själv hantera programmets variabler, tabelluppslagning, subrutinernas argument och så vidare. Du kan även göra en annan uppdelning än pseudokoden ovan så länge du inte bryter mot det givna programflödet.

Internationella morsealfabetet och binärkodning i laborationen

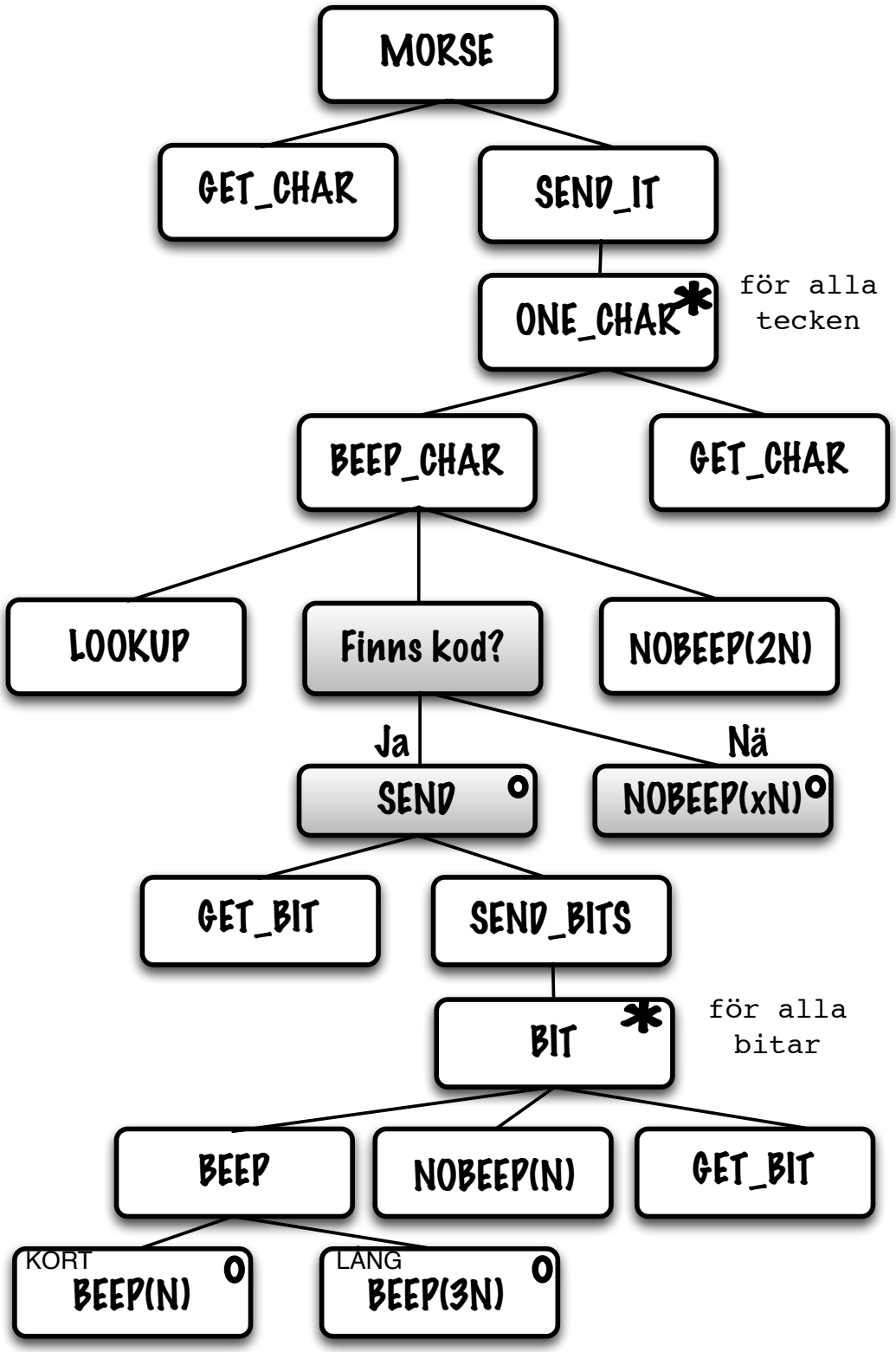
Använd tabellinnehållet för `BTAB`. Notera att ASCII-tecknet för mellanslag `$20` saknas i tabellen. Detta faktum hanteras i **uppgift 2**.

Programmet i laborationen skall kunna hantera samtliga bokstäver i tabellen.

Tkn	ASCII	Morsekod	Binärkod	Hex
A	$41	.-	01100000	$60
B	$42	-...	10001000	$88
C	$43	-.-.	10101000	$A8
D	$44	-..	10010000	$90
E	$45	.	01000000	$40
F	$46	..-.	00101000	$28
G	$47	--.	11010000	$D0
H	$48	....	00001000	$08
I	$49	..	00100000	$20
J	$4A	.---	01111000	$78
K	$4B	-.-	10110000	$B0
L	$4C	.-..	01001000	$48
M	$4D	--	11100000	$E0
N	$4E	-.	10100000	$A0
O	$4F	---	11110000	$F0
P	$50	.--.	01101000	$68
Q	$51	--.-	11011000	$D8
R	$52	.-.	01010000	$50
S	$53	...	00010000	$10
T	$54	-	11000000	$C0
U	$55	..-	00110000	$30
V	$56	...-	00011000	$18
W	$57	.--	01110000	$70
X	$58	-..-	10011000	$98
Y	$59	-.--	10111000	$B8
Z	$5A	--..	11001000	$C8

Strukturdiagram för uppgift 3

Strukturdiagrammet nedan är modifierat enligt uppgift 3. De tillkomna rutorna är gråade.
Vad skall `x` vara i `NOBEEP(xN)`?



-o-O-o-