Kort beskrivning av Sveriges första dator BESK

Bo Einarsson
MAI
10 januari 2005

Den första svenska egentliga datorn är BESK, vilket står för Binär Elektronisk SekvensKalkylator. Den var körklar den 6 november 1953 och togs i reguljär drift under februari 1954.

Räknearbeten som tog en människa fem dagar tog på företrädaren BARK (Binär Automatisk ReläKalkylator) två timmar och på BESK endast en minut.

BESK hjälpte till att spå väder, lösa hållfasthetsproblem åt Vattenfall, generera tabeller åt försäkringsbolag, undersöka vibrationer åt ASEA, osv. Dessa är uppgifter som fortfarande utförs av superdatorer, men när BESK var klar trodde man att dess kapacitet skulle räcka för Sveriges behov.

Tekniska data för BESK

Primärminne (ferritminne): 1024 helord om 40 bitar, dvs 5 KiB

Sekundärminne (trumminne): 8192 helord om 40 bitar, lagrade i 256 kanaler om 32 helord, dvs 40 KiB

Talrepresentation: Binär med helord x i intervallet

$$-1 \leq x < 1$$

Additionstid: 42 μs
Multiplikationstid: 350 μs
Divisionstid: 686 μs
Ungefärlig hastighet: 5000 FIOPS
(FIOPS = fixpunktsoperationer per sekund)
Ungefärlig hastighet: 250 FLOPS
(FLOPS = flytpunktsoperationer per sekund)

Jämförelse:
Monolith vid NSC klarar 1 132 000 000 000 FLOPS eller drygt 1 TFLOPS (november 2002). IBM Blue Gene klarar drygt 70 TFLOPS (november 2004).

Inmatning: Pappersremsa 400 tecken (hexadecimala tal) i sekunden

Utmatning: Pappersremsa 145 tecken i sekunden eller skrivmaskin 15 tecken i sekunden

Anm: Under denna tid användes i Stockholm den korrekta benämningen *sedecimala tal* för vad som numera kallas *hexadecimala tal*.

Notis: BESK och dess närmaste efterföljare SARA och FACIT EDB-3 hade alla den egenheten att $(-1)\times(-1)$ blev -1, medan alla andra multiplikationer som $(-0{,}5)\times(-0{,}5) = 0{,}25$ blev korrekta. Notera att talet $+1$ inte var ett tillåtet normaliserat tal.

Programmering av BESK

Programmering av BESK skedde helt hexadecimalt, både för operationer och adresser. Varken memotekniska namn på operationer (“ADD” i stället för “10”) eller namn på variabler fanns att tillgå.

I ett helord med dess 40 bitar ryms 10 hexadecimala siffror. Man använder sig mycket av halvord om 20 bitar och således 5 hexadecimala siffror. De tre första användes för adressen, som går från 000 till 7FF (decimalt 2047), de två sista för operationen. Grundfomen för addition är “10”, dessutom finns tre tilläggsformer enligt nedan:

Ordern 15810 innebär att vänster halvord i position 158 adderas till ackumulatorregistret, med resultatet i ackumulatorregistret.
Ordern 15830 innebär att helordet i position 158 adderas till ackumulatorregistret, med resultatet i ackumulatorregistret.
Ordern 15850 innebär att vänster halvord i position 158 kopieras till ackumulatorregistret.
Ordern 15870 innebär att helordet i position 158 kopieras till ackumulatorregistret.

Ordern 15970 uppfattas som ett fel, en udda address kan inte svara mot ett helord, maskinen stannar! Om man då trycker på start utförs operationen som en halvordsoperation, dvs 15950.

Den mer eller mindre fiktiva adressen 800 utnyttjas som startpunkt för subrutiner, som systemet sedan lägger in på en lämplig adress.

Exempel 1: Här kommer en programsnutt för att utföra beräkningen $x+y-z$. Talen finns i helorden med adress 100, 102 och 104, resultatet önskas i det helord som har adressen 180, nedan kallat cell 180.

10070 x till AR
10230 $x + y$ till AR
1042B $x + y - z$ till AR
18031 $x + y - z$ till cell 180

Exempel 2: Här kommer en programsnutt för att byta tecken på innehållet x i ackumulatorregistret. Vi kopierar först värdet till cell 106.

10631 x till cell 106
1066B $-x$ till AR

Wheeler-hopp

Ibland är det önskvärt att i ett program bryta den vanliga sekventiella exekveringen och hoppa till ett annat ställe i programmet. Den vanliga hoppsatsen kan då användas; men ibland skall samma följd av instruktioner utföras flera gånger på olika ställen i programmet. Det är då olämpligt att behöva upprepa dessa satser. En möjlighet är då att använda sig av ett Wheeler-hopp*.

Antag att prgramavsnitten A och B skall utföras flera gånger. Det är då tillräckligt att att avsnitten finns vardera en gång i programmet, och att man hoppar till respektive avsnitt, men sedan måste man hoppa tillbaks till rätt ställe. Det är detta senare problem som Wheeler löst.

*David John Wheeler (1927 - 2004) utvecklade “Wheeler Jump” för att tillåta ett program att flytta exekveringen till en subrutin, en föregångare till “goto” satsen som är välkänd för alla som programmerat i Basic. Hans erfarenhet att skriva program för EDSAC i Cambridge från mars 1949 ledde Wheeler och hans kolleger Maurice Wilkes och Stanley Gill till att skriva den första boken om programmering: “The Preparation of Programs for an Electronic Digital Computer”, 1951.

Antag att man lämnar huvudprogrammet vid order “n”, där “n” är jämnt, och att “b” är startorder till subsekensen. Man skriver i huvudprogrammet

```
rad      order
---      ------
n-1      n-1 50
n        b   0C
```

Order “n-1”adderar högerhalvordet “n-1” (där ordern själv står) till ackumulatorregistret efter dess nollställning. I detta register står då

```
00000 n-1 50
```

Här representerar “n-1” de tre hexadecimala siffrorna i adressdelen. Härefter utföres hoppet “b 0C”, vilket ej påverkar ackumulatorregistret.

Första orderna i subsekvensen är

```
rad       order
---       ------
b         001 10
b+1       u   07
```

I order “b” adderas talet 2 till adressdelen i ackumulatorregistret, som således blir

```
00000 n+1 50
```

Adressen “u” i nästa order anger platsen för den hopporder varifrån återhoppet sker, Order “u” är alltid udda. Order “b+1” kommer alltså att sätta “n+1” i adressdelen i order “u”. Återhoppet sker alltså till order “n+1” i huvudprogrammet.

Flyttalsaritmetik på BESK

Som tidigare nämnts är BESK konstruerad för fixpunktsaritmetik med alla tal mellan -1 och 1, nedre gränsen tillåten, övre gränsen ej tillåten.

Programmeraren var därför tvungen att skala alla tal, om resultatet av en beräkning blev till beloppet mindre än 0,5 fanns risk för extra noggrannhetsförlust, om det blev till beloppet större än 1 inträffade “spill”, vilket fick åtgärdas med omnormalisering.

För att förenkla användningen av BESK utvecklade Matematikmaskinnämnden under hösten 1957 FLINTA, ett interpretativt system för räkning med flytande binärpunkt (dvs flyttalsaritmetik).

Flyttalen lagrades på ett sätt som ganska liknar IEEE 754, den normerade taldelen fanns i position 0-31 och en med det hexadecimala talet 80 (decimalt 128) förskjuten exponent i position 32-39. Den första positionen (0) är taldelens tecken, negativa taldelar representeras av sitt komplement. Talomfånget blev från $1{,}4 \cdot 10^{-48}$ via det minsta normaliserade $1{,}5 \cdot 10^{-39}$ till $1{,}7 \cdot 10^{38}$.

Följande är citat ur skriften SNABBKODSSYSTEMET FLINTA: Operationerna tar relativt lång tid, de flesta tar mellan 1 och 3 millisekunder. Om Flinta användes för kodning av ett problem med lång maskintid kan det bli nödvändigt att koda lämpliga delar av programmet i vanlig maskinkod för att minska räknetiden.