Övning1 Datorteknik, HH vt12 - Talsystem, logik, minne, instruktioner, assembler

Talsystem

Talsystem – binära tal

F1.1) 2^n stycken tal från 0 till 2^n-1

F1.2) 9 bitar (512 kombinationer)

Talsystem – 2-komplement

F1.3) talet 7 = 0111 talet –7 = 1001

F1.4) största (positiva) 127, mest negativa –128

F1.5)

a) –6
b) 90
c) –2
d) 14803

F1.6)

a) 01100110
b) 01000000
c) 00101100
d) 10000000
e) 01111111

F1.7)

a) 11111010
b) 00011001
c) 11111000
d) 00000001

F1.8) 493F

F1.9) unsigned tal från 0 till 2^n-1
signed -2^n till 2^n-1

Talsystem – Binär aritmetik. Flaggor

F1.10)

a) 1100
b) 1010
c) 1111
d) 01011
e) 10000

F1.11) Multiplikation med 2

F1.12)

a) 1100 (binary) or −4 (decimal)
b) 01010100 (binary) or 84 (decimal)
c) 0011 (binary) or 3 (decimal)
d) 11 (binary) or −1 (decimal)

F1.13.a)

```
    1101
+   1000
-------
   10101
```

N=0 Z=0 V=1 C=1 Minnessiffra utanför 4 bitar. Overflow: Två negativa tal adderas och resultatet blir postivt (bit3)

F1.13.b)

```
    0110
+   0101
-------
     1011
```

N=1 Z=0 V=1 C=0 Overflow: Två positiva adderas till ett negativt. (bit3 i termerna = 0 och bit3 i svaret = l). Resultatet negativt, dvs bit3 i svaret = 1.

F1.13.c)

```
+
    0111
+   1001
-------
   10000
```

N=0 Z=1 V=0 C=1 4-bitarstalet är noll, En bit utanför (dvs Carry)

F1.13.d)

```
    0110
+   1001
-------
    1111
```

N=1 Z=0 V=0 C=0 Negativt tal, dvs högsta av 4 bitar = l

Talsystem - Flyttal

F1.14)
S =0, dvs positivt tal!
E= 10000011, dvs talet 128 + 2 + 1 = 131 => $2^{131-127} = 2^4$
M = 01111010100000000000000
101111010100000000000000 * 2^4 = 10111.1010100000000000000

= $2^4+2^2+2^1+2^0+2^{-1}+2^{-3}+2^{-5}$ = 23 + 0.5 + 0.125 + 0.03125 = 23.65625

F.1.15)
2.150 = 2 + 0.150
2 = 10b
0.150:
0.150*2 = 0.3 0
0.3*2 = 0.6 0
0.6*2 = 1.2 1
0.2*2 = **0.4** 0
0.4*2 = 0.8 0
0.8*2 = 1.6 1
0.6*2 = 1.2 1
0.2*2 = **0.4** 0
stannar här pga av upprepning. 0011 kommer upprepa sig. Annars stannar man när det blir noll!

Talet som ska lagras = 10.001**0011** <- upprepning av fyra sista, dvs
10.00100110011001100110011

Lagring:
Teckenbit 0 (positivt)
Mantissa: Talet normaliserar så det bara blir en etta innan decimalpunkten dvs.
10.00100110011001100110011 = 1.000100110011001100110011 *2^1
Mantissa = 00010011001100110011001100 (23 bitar av talet)
Exponent= 1
Enligt standard ska E-127 = 1 => E=128
Svaret alltså

0	10000000	00010011001100110011001

Minne

Minne – Grunder

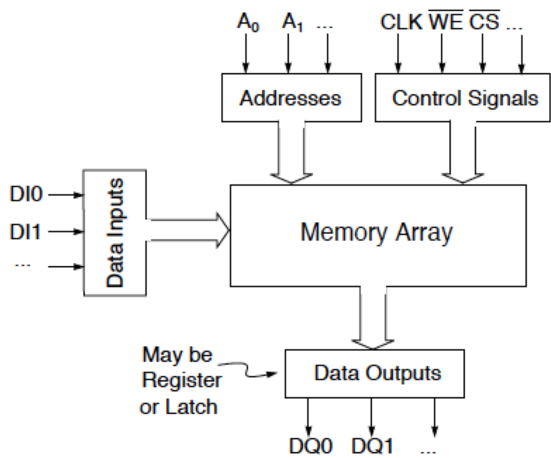
Notera att 1 byte = 8 bitar
1k = 1024 = 2^{10}
1M = 1024*1024 = 2^{20}
1G = 1024*1024*1024 = 2^{30}

F2.17. Att man har 2^{15} (32768) byte i minnet. Varje plats är på 16 bitar, dvs 2 byte. Alltså blir det 2^{14} unika 16-bitars positioner.
F2.18. 16 pinnar. Vi skulle anta 8-bitars databuss. 64kB = 2^{16} positioner, dvs det krävs 16 bitar för att nå så många positioner.

F2.19. Minnet är organiserat som rader och kolonner. Man kan säga att man med adressen väljer ut vilken rad man vill nå. Sen beroende på vilka transistorer på den raden som "leder" resp "inte leder" så får man etta/nolla. Kolonnerna motsvarar sedan min information på just den adressen.

F2.20. 32 bitars adressbuss = 2^{32} unika positioner. Varje position Innehåller 16 bitar= 16*2^{32} bitar stort minne. 1 byte = 8 bitar=> 8589934592 byte stort. = 8Gbyte

F2.21. 4 databen = 4 bitars ordlängd;
14 adresspinnar = 2^{14} = 16384 positioner = dess adressrymd;
16k positioner á 4 bitar = 8 kByte



Blockdiagram av ett synkront SRAM

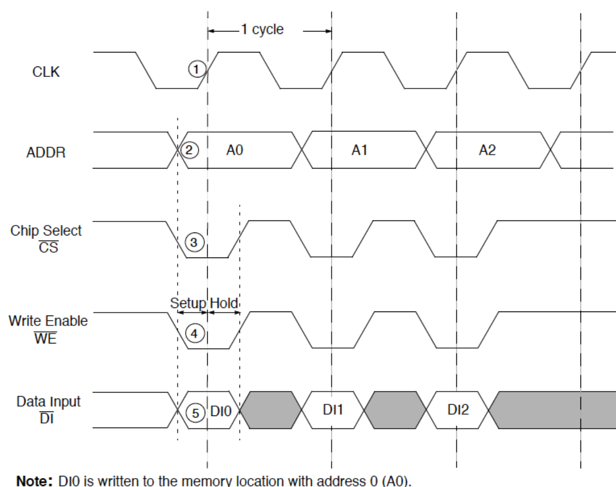
Minne- Minnestyper

F2.22.

RAM	Random Access Memory. Datorns vanliga primärminene. Kräver spänning på för att lagra information. Alla celler tar lika lång tid att nå. Läs och skrivbart.
ROM	Read Only Memory. Endast Läsbart minne. Skrivs av tillverkaren
SRAM	Static RAM. Latchat minne där varje cell i stort sett är en D-vippa. Kräver 6 transistorer/bit, men är statiskt så länge ström finns.
DRAM	Dynamiskt RAM. Entransistorlösning. Nyttjar strökapacitansen inuti transistorn för lagring av information. Kräver Refresh, dvs att man hela tiden läser av minnet samt skriver tillbaka informationen. Detta pga av urladdning av kondensatorn. Mindre minne, men långsammare än SRAM pga refresh. Idag sker refresh ofta internt i minnena
EPROM	Erasabale Programmable ROM. Skrivbart ROM. Skrivning sker genom att man kan manipulera gatespänningen (sk. floating gate) på transistorerna så att dom leder resp. inte leder då dom adresseras. Görs med högre spänning. Entransistorlösning. Raderas av UV-Ijusbestrålning ca 20min.
EEPROM	Electrical Erasabale Programmable ROM. Skrivbart ROM. Skrivning sker genom ökad spänning. Samma princip som för EPROM, men kräver två transistorer. Kan raderas elektriskt på byte-nivå, dvs 8 bitar åt gången.
FLASH	Nyare variant av EEPROM. Entransistorlösning. Går att radera hela block av celler på en gång.

	Fördel	Nackdel
SRAM	Snabbt	Stort, kräver 6 transistorer
DRAM	Litet	Långsammare, komplexare pga refresh

F2.24. Förklara hur en skrivning går till från minnet. Ange speciellt vad som sker på de olika bussarna.



Skrivning till minnet (ifrån CPUn)

Datormodell

F2.1) Minne, CPU, I/O, Styrenhet

F2.2) Processorn kommunicerar med minnet via två register MDR (Memory Data Register) och MAR (Memory Data Register).

skrivning:

1. Adressen till det som ska skrivas läggs i MAR
2. Data som ska skrivas läggs i MDR
3. skrivning!

läsning:

1. Adressen till det som ska läsas läggs i MAR
2. Läsning!
3. Data som lästs ut från minnet läggs i MDR

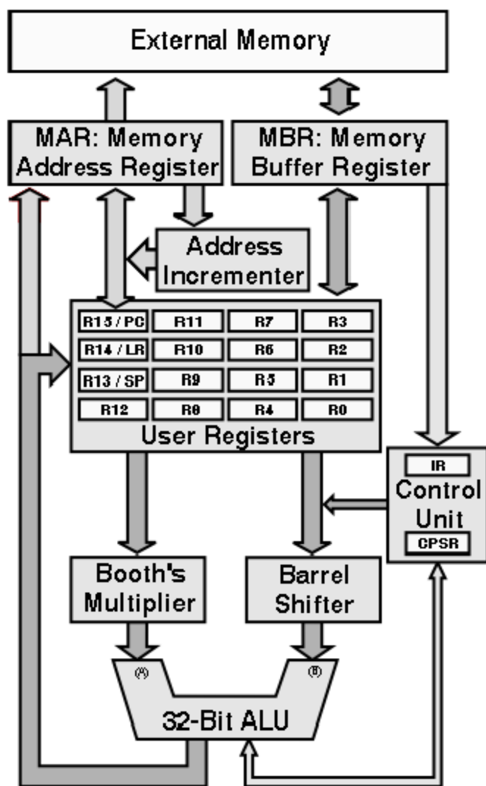


Figure 3.1: ARM Block Diagram

F2.3

Det går inte att avgöra! Det är upp till användaren att se till så att talet inte körs om det skulle vara ett TAL. Datorn kan gladeligen försöka köra talet som en instruktion.

F2.4

Styrenheten är den enhet som kodar av instruktionen och avgör vilka andra delar i datorn som ska nås och vid vilka tidpunkter. Det är den som styr alla de andra enheterna (CPU, MINNE osv)

F2.5

11 bitars adresslängd, 8 bitars ordlängd = 2^{11} positioner á 1 byte= 2kByte

om man använder 2 bitar i PCLATH-registret för att ange sidan får man 2^2 = 4 sidor där man på varje sida kan nå 2kByte = 8kByte totalt

Instruktionsbegreppet

F2.6)

27 opcoder: Går åt 5 bitar för att representera dessa

16 register: Går åt 4 bitar för att välja ut register

Alltså:

OPCODE (5)	SR (4)	DR (4)	IMM (32-5-4-4 = 19)

IMM är på 19 bitar. största resp minsta tvåkomplementtal man kan representera med 19 bitar =

Max: 2^{19-1}-1 = 262143	Min: -2^{19-1} = -262144

F2.7) Minnet, register samt del av instruktion (immediate)

Instruktionscykel

F2.8)

a) `LDR R5,[R6,#5]`

Instruktionscykeln

FETCH	Hämta instruktionen från minnet
DECODE	Avgör att det är en LDR-instruktion
EVALUATE ADRESS	Beräkna adress R6 + 5
FETCH OPERANDS	Hämta data från adress R6+5
(EXECUTE)	(ej för LDR, inget som beräknas)
WRITE BACK	Lägg minnesinnehållet i R5

b) `SUB R1, R2, R4`

Instruktionscykeln:

FETCH	Hämta instruktionen från minnet
DECODE	Avgör att det är en SUB-instruktion
(EVALUATE ADRESS)	görs ej. Inga data i adresser
FETCH OPERANDS	Hämta data R2 och R4 och förbered för ALU
EXECUTE	Beräkna R2-R4, (R2 + 2-komplementet av R4)
WRITE BACK	Lägg resultatet i R1

c) `B TEST`

Instruktionscykeln:

FETCH	Hämta instruktionen från minnet
DECODE	Avgör att det är en Hopp-instruktion
EVALUATE ADRESS	Beräkna adress att hoppa till. (PC+tal)
(FETCH OPERANDS)	-
EXECUTE	PC = beräknad adress
WRITE BACK	-

F2.9) Adresseringsmode är namn på de olika sätt instruktioner kan nå data

F2.10) Förklara följande olika adresseringsmode

a) Immediate - Operanden ligger direkt i instruktionen
b) Direct - Adressen till operanden ligger direkt i instruktionen
c) Base+Index - Adressen till operanden fås genom [basregister +värde]
d) Register - Alla operander i register

F2.11) Vilka adresseringsmode används i följande fall:

a) *Register*, All data finns i register
b) *Direct*, Data finns på en adress som anges direkt i instruktionen
c) *Base +index*, Data finns på en adress som beräknas med basregister + indexvärde
d) *Immediate*, Själva datat finns direkt i instruktionen (inte i minne eller register)

F2.12)

a) *Immediate* Instruktion där data finns i instruktionen

```
ADD R1,R1,#5
```

b) *Direct* Adress till data står i instruktionen

```
LDR R2, ADRESS
```

c) *Base+Index* Basadressindexering

```
LDR R2, [R4,#5]
STR R2, [R4]
```

d) *Auto+index* Basadressindexering + uppdatering

```
LDR R2, [R4],#5
LDR R2, [R4,#5]!
```

e) *Register* Data i register

```
ADD R2, R3, R4
```

Assembler

F3.13

Direktiv styr översättaren (assemblatorn). Ingen instruktion för processorn
Instruktioner är det som processorn utför

F3.14

Att man översätter koden i två steg. Första steget beräknar man alla adresser som labels i koden har. I Andra genomgången översätts själva instruktionerna.

F3.15

Label är ett namn istället för en adress. Gör kod lättare att skriva och man behöver inte några absoluta adresser i koden. Gör koden mer flexibel. Mycket lättare att ändra

F3.16 Symboltabell är den tabell över labels som assemblatorn gör i första steget i assembleringen

Programmering

Notera att programmeringslösningarna endast är förslag. Som vanligt med programmeringsuppgifter kan man göra på flera olika sätt!

F3.Pla

```
            CMP Rl, R2
            BHI rl_big ; Branch if higher, gäller unsigned
            MOV R3, R2
            B   cont
rl_big      MOV R3, Rl
cont        B   cont    ; Avsluta med evig loop
```

F3.Plb

```
            CMP Rl, R2
            BGT rl_big  ; Branch if Greater Then. Gäller signed
            MOV R3, R2
            B   cont
rl_big      MOV R3, R l
cont        B   cont    ; Avsluta med evig loop
```

F3.P2

```
            LDR RO, =ADRESS
            LDR Rl, =DEST
            LDR R3, [RO]
            STR R3, [Rl]
STOP        B STOP      ; Avsluta med evig loop
ADRESS      DCD 0x40000000
DEST        DCD 0x20000000
```

F3.P3

```
            ; Anta adress till talen ligger i R3
            MOV R4, #0
Next        LDRb RO, [R3], #2 ; läs tal. Rl=mem[R3] R3=R3+2
            CMP RO, #0
            BEQ STOP                ; om 0 => avbryt
            ADD R4, R4, R0          ; Summera upp i R4
            B Next
STOP        B STOP
```

F3.P4

```
            LDR Rl, =43
            LDR R2, =PORTO
            STRb Rl, [R2]
STOP        B STOP
```

F3.P5

```
            LDR  R2, =PORTO
            LDRb Rl, [R2]
            AND  Rl, Rl, #0x04 ; Bitmaska för bit3
            CMP  Rl, #0        ; Blev det noll var bit3=0
            BEQ  load0
            LDR  Rl, =0xFF
            B    store
Load0       MOV  R1, #0
Store       STRb R1, [R2]
STOP        B    STOP
```

F3.P6

```
            LDR  R3, =0xF0000000
            AND  R2,R1,R3
STOP        B STOP
```