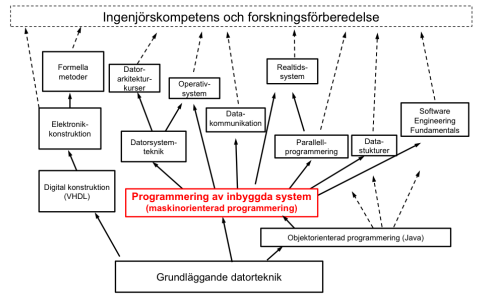
Syften och målsättningar

Syften och målsättningar

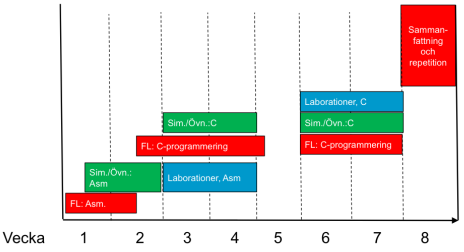
- Kursens syften är
  - att vara en introduktion till konstruktion av små inbyggda system och
  - att ge en förståelse för hur imperativa styrstrukturer översätts till assembler
  - att ge en förståelse för de svårigheter som uppstår vid programmering av händelsestyrda system med flera indatakällor.
- Centrala målsättningar är att kunna:
  - skriva enkla C-program med användande av programspråkets datatyper och styrstrukturer
  - beskriva motsvarigheten i assembler till typiska programstrukturer i C.
  - utnyttja de i kursen använda verktygen för programutveckling på ett adekvat sätt
  - medverka vid konstruktion och programmering av enkla inbyggda system med givna komponenter
  - konstruera system innefattande olika typer av undantag (interna undantag, avbrott, återstart)
  - beskriva och exemplifiera några olika typer av digitala kringkomponenter och deras användning.

CHALMERS Programmering av inbyggda system



CHALMERS Programmering av inbyggda system

Genomförande



CHALMERS Programmering av inbyggda system

Laborationsöversikt

- Programutveckling i assembler
  Moment 1: ”Övervakning/styrning av borrmaskin”
  Moment 2: ”Pseudoparallell exekvering”
- Programutveckling i C
  Moment 3: ”Länkad lista med pekare”
- Maskinnära programmering i C
  Moment 4: ”Övervakning/styrning av borrmaskin”

CHALMERS Programmering av inbyggda system

Inför laborationerna

- Laborationerna måste vara väl förberedda innan laborationstillfället
- Utveckling och test kan göras med simulatorer
- Använd kodnings-/simuleringsövningar och hemarbete för förberedelserna
- ETERM, CodeLite och XCC12 finns på kursens ”resurssida”, hämta och installera omgående
- OBS: Laborationerna börjar Måndag i läsvecka 3
  ANMÄL ER OMGÅENDE (via kursens hemsida)

Kursen börjar.

ISA är programmerarens bild av microprocessorn. Tänk dig ett API men ändå inte ;P

I ISA ingår de olika punkterna till höger. Exempel på instruktionsgrupp är `Aritmetik och Logik` och `Branches`.

Korttidslagring kan använda sig av till exempel registerlagring. Registerlagring är det allra närmaste processorn och vi snackar pikosekunder.

Speciella register används för att hålla statusar som flaggor eller interuptstatusar.

Datatyper och Storlek

68HCS12

Blockdiagram

En fysisk pinne får ett eget namn och kan rotas om till en specifik funktion. Man kan kommunicera till andra enheter via trådar.

Detta bidrar till en programmerbar enhet där till och med arbetstakten är programmerbar.

Bankswitching i hårdvaran innebär att man kan mappa in olika delar av hårdvaran varför det finns hela 256kb minne till en 16bitars addressbuss.

Instruktionsuppsättning

”ISA” – Instruction Set Architecture

- Vilka operationer kan utföras ?
  - Instruktionsgrupper
- Hur lagras operanderna förutom i minnet ?
  - Korttidslagring
- Hur nås operander i minnet?
  - Adresseringssätt
- Vilka typer/storlekar av operander kan hanteras ?
  - Generella/speciella register, registerstorlek

Programmerarens bild – datatyper/storlek

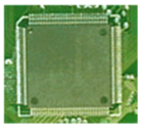
	char (8)	short int (16)	long int (32)	long int (64)	floating point (IEEE)	pointers
68HCS12	X	X				16/20 bit
Coldfire V1	X	X	X			32 bit
Coldfire V4	X	X	X		X	32 bit
PowerPC	X	X	X		X	32 bit
PowerPC (64)	X	X		X	X	64 bit
8086	X	X				16/20 bit
80386	X	X	X			32 bit
80486	X	X	X		X	32 bit
X86-32	X	X	X		X	32 bit
X86-64	X	X		X	X	64 bit

Programmerarens bild – adresserbart minne

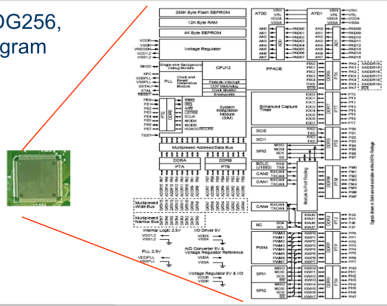
ADRESSBUSS	RANDOM ACCESS
16 bitar	2^{16} = 65 536 byte = 64 kbyte
20 bitar	2^{20} = 1 048 576 byte = 1 024 kbyte = 1 Mbyte
24 bitar	2^{24} = 16 777 216 byte = 16 384 kbyte = 16 MByte
32 bitar	2^{32} = 4 294 967 296 byte = 4 194 304 kbyte = 4 096 Mbyte = 4 Gbyte
64 bitar	2^{64} = 1,844674407 10^{19} byte = 16 Ebyte

Freescale 68HCS12

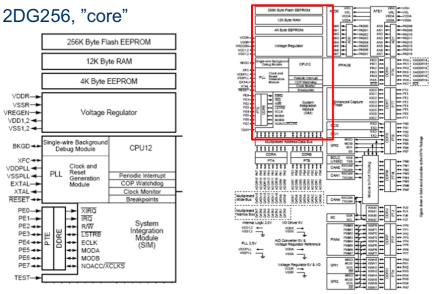
- HCS12 adressrum, IO och minne
- CPU12, klockor och räknare
- "Random Access"- Minne
  - RWM, FLASH, EEPROM
- Periferienheter
  - Parallell Input/Output:
  - Seriell kommunikation
  - AD
  - PWM



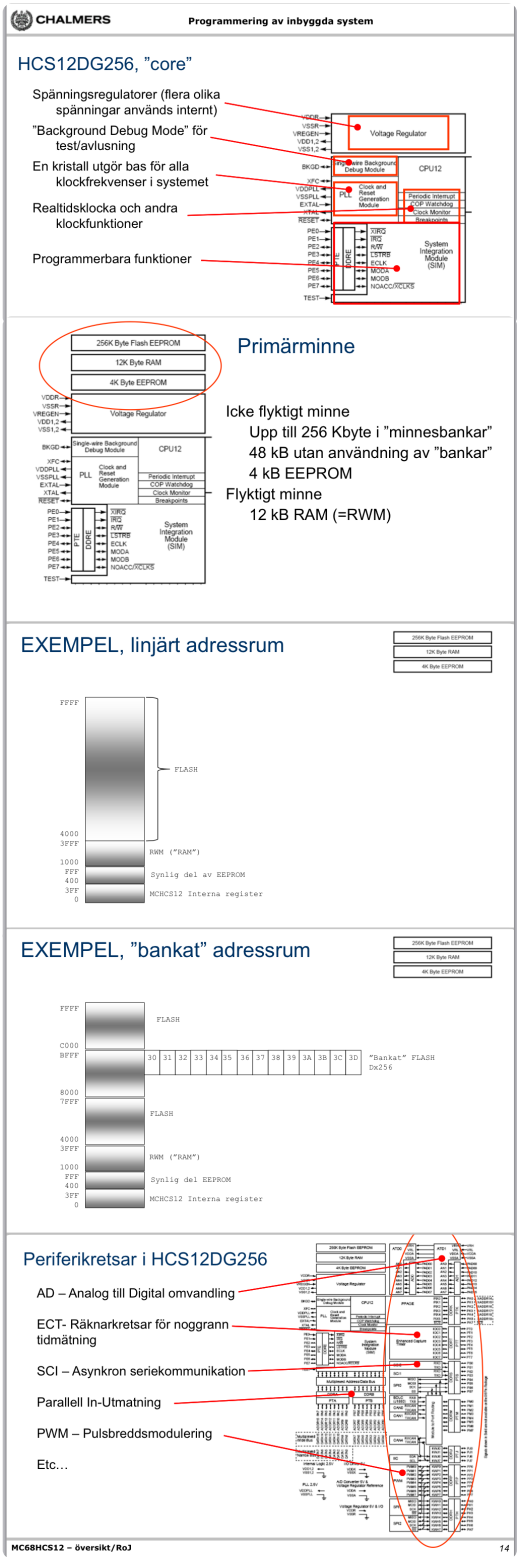
HCS12DG256, blockdiagram



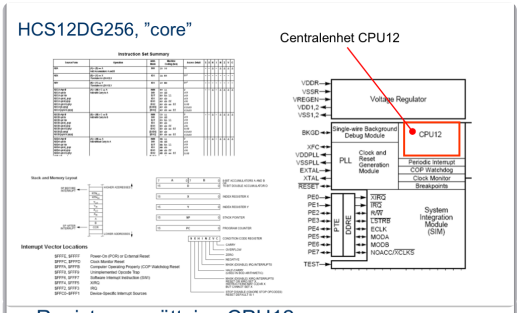
HCS12DG256, "core"



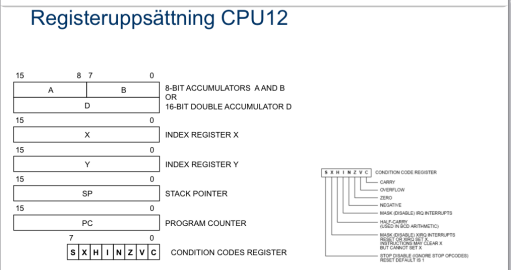
När CPUn vill ha en funktion från en bank så skriver den det till systemet varpå banken "switchas".



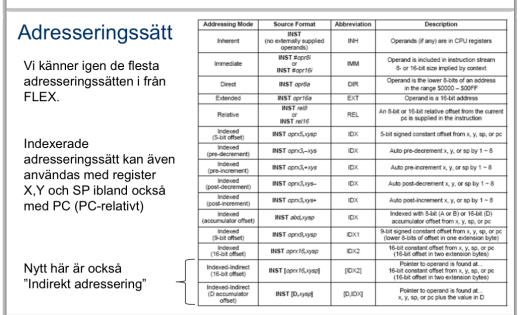
Vi är tillbaka på `programmerarens bild` och detta kommer vi behandla sist i kursen.



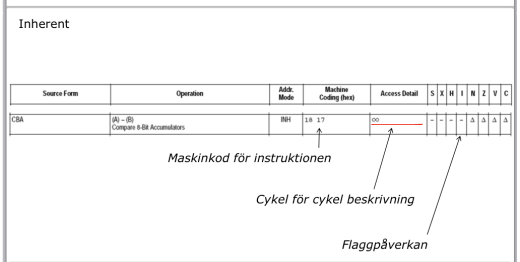
CPU12 är kärnan medan HCS12 är hela kretsen.
En arkitektur lik denna kallas `8/16-bitars arkitektur` och detta är beteckningen vilken används i kurslitteraturen.



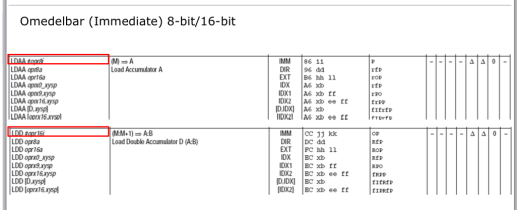
Indexed är allt som har att göra med `X,Y` och liknande.
Nytt här är `Indirekt adressering` indexerade anrop används vid skapande av tabeller och massa annat bös. Det är lite överflödigt och de är inte jätteintressanta. Man klarar sig gott på Flispens addresseringssätt.



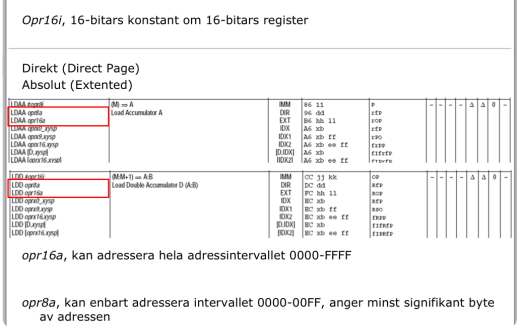
Inherent
Antalet bokstäver talar om hur många cykler det tar.



Immediate



Direct Page
Kan bara adressera de första 256 byten i minnet.



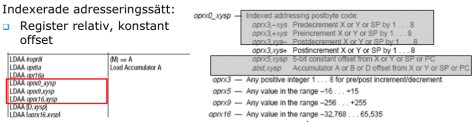
Relative (Branch)

PC-relativ (”BRANCH”-instruktioner)
- 8-bitars offset (-128..127)
- 9-bitars offset (-256..255)
- 16-bitars offset (-32768..32767)

BRA *rel8*	Branch Always (if 1 = 1)	REL	20 rr
IBEQ *abdxys, rel9*	(cntr) + 1→ cntr If (cntr) = 0, then Branch else Continue to next instruction Increment Counter and Branch if ≠ 0 (cntr = A, B, D, X, Y, or SP)	REL (9-bit)	04 1b rr
IBNE *abdxys, rel9*	(cntr) + 1→ cntr if (cntr) not = 0, then Branch; else Continue to next instruction Increment Counter and Branch if ≠ 0 (cntr = A, B, D, X, Y, or SP)	REL (9-bit)	04 1b rr
LBCC *rel16*	Long Branch if Carry Clear (if C = 0)	REL	10 24 qq rr

Indexed

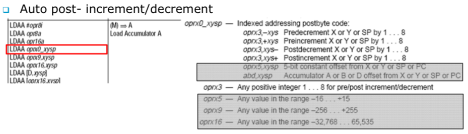
Indexerade adresseringssätt:
- Register relativ, konstant offset



Basregister kan vara något av: X,Y,SP,PC
EXEMPEL:
```
LDAA   5,X
STAA   20,Y
LDAA   sym,PC
STA    off,SP
...
```
Observera, ingen syntaktisk skillnad.
Assemblator väljer effektivast kodning

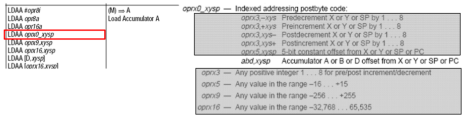
Specialfall: `n,PCR`
```
LDAA   sym,PCR
```
Antag `PC` pekar på nästa instruktion.
Operanden är här `PC-sym`, jfr offsetberäkning för "BRA"-instruktioner

Indexerade adresseringssätt:
- Auto pre- increment/decrement
- Auto post- increment/decrement



Basregister kan vara något av: X,Y,SP
EXEMPEL:
```
LDAA   1,-X
STAA   4,Y-
STAB   8,+SP
LDAB   7,SP+
...
```

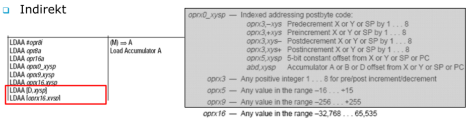
Indexerade adresseringssätt:
- Register relativ, offset i ackumulator



Basregister kan vara något av: X,Y,SP,PC
EXEMPEL:
```
LDAA   A,X
STAA   B,Y
STAB   D,SP
LDAB   D,PC
...
```

Hör pekas det på en minnesaddress vilken i sin tur innehåller data istället för att peka direkt på data.

Indexerade adresseringssätt:
- Indirekt



EXEMPEL:
```
LDAA   [D,X]
STAA   [sym,PCR]
STAB   [2,SP]
LDAB   [D,Y]
...
```

MC68HCS12 – översikt/RoJ 25

Efter detta kom massa instruktioner vilka ändå är helt onödiga att lära sig utan till. Kolla i litteraturen för att få veta hur du gör för att flytta. Väl värt att nämna är dock att man bara ska använda ett fåtal av alla instruktioner. Se sida 28-ish i .pdfn. Man klarar sig gott med kortvarianten vilken finns som pdf. Vill man ha en lite utökad version finns det en med detaljerade instruktioner på Cremona.
*Precis som med kurslitteraturen får man ha med sig **en av** listorna till tentan.*

Logiska skift används för tal *med* tecken. Aritmetiska skift används för tal ***utan*** tecken. Det är bara högerskiften samt flaggsättningen som skiljer sig åt mellan de båda.