Introduktion til programmering

Af mikroprocessor Atmel ATmega328P i en Arduino Uno

Min baggrund:

Intel 4004, 4 bit, maskinsprog
Intel 8008, 8 bit, maskinsprog bit for bit

I sprogene: assembler, Fortran IV, Basic , Pascal på processorerne
Intel 8080, 8085, 8039, 8051, 8088,
samt Varian Data Machines og Digital Equipment og endelig
Dansk Data Elektronik (8085 + 8”drev + IEC625/IEEE488 busser)

I sprogene C og C++ på PC.

De to maskiner Varian og Digital var til tekniske opgaver.
de er begge ret ukendte udenfor tekniske kredse.
16 bit maskiner bygget af S og H kredse,
Kernelager, hulstrimmel.
Max lager 64 k x 16 bit

Varian og jeg leverede de første gammakameraer i Danmark,
Efterfølgerne CT og MR scannere, og elektroencefalografer er
videreudviklinger heraf.

Nogle Tal definitioner:

Nibble: Primært er det en brystvorte,
sekundært er det et 4 bit bredt ord
Mine lærlinge har altid fået lov at berøre nibblerne.

Byte: Et 8 bit bredt ord.

Word: Et 16 bit bredt ord.

Integer: Heltal, mindste værdi i 8 bits bredde = 0 til 256,
MEN ingen øvre grænse bare vi har bitbredde nok.

Real: Består af en heltalsdel og af en eksponent

Floating Point: Er et kommatal, som Real, men med en kommaplacering

Talsystemer

1. Binært talsystem:

    Base er 2, d.v.s. værdien for hvert bit
    kan være 0 eller 1, eller anderledes udtryk: Sand/Falsk
    Værdien afhænger af positionen.
    Forstås af computere, er umenneskeligt sprog

2. Måden vi anskuer et binært ord på:

    A: Hex, en halv byte = en nibble = 4 bit, 0 til F
    B: Hex men 8 bit, værdier fra 0 til FF
    C: Octal, 3 bit ad gangen, værdier fra 0 til 7

Et enkelt eksempel på en binær byte:
1000 0010 = 82 HEX
10 000 010 = 202 Octal, MEN vi mener det samme!

Hex og octal er altså kun en humaniseret anskuelse.

En anskuelse af binære tal.

Hex	Decimal	Binær	Octal
00H	0	0000	0 000
01H	1	0001	0 001 0 1 octal
02H	2	0010	0 010 0 2 octal
03H	3	0011	0 011 0 3 octal
04H	4	0100	0 100 0 4 octal
06H	6	0110	0 110 0 6 octal
07H	7	0111	0 111 0 7 octal
08H	8	1000	1 000 1 0 octal
09H	9	1001	1 001 1 1 octal
0AH	10	1010	1 010 1 2 octal
0BH	11	1011	1 011 1 3 octal
0CH	12	1100	1 100 1 4 octal
0DH	13	1101	1 101 1 5 octal
0EH	14	1110	1 110 1 6 octal
0FH	15	1111	1 111 1 7 octal
5 taller mangler for at se om i var vågne!			

Filtyper og programmer i programmering

1. Kildetekst, kaldet sourcekode:
   Det du skriver i en editor, IKKE en tekstbehandler

2. Binær kode, ofte kaldet og betragtet som HEX:
   Det underliggende binære tal din computer – måske -- kan forstå

3. Biblioteker:
   Samling af programstumper,
   som gør livet lettere for os alle.
   Findes som kildetekst og som Runtime biblioteker

4. Linker:
   Det hjælpeprogram, som sammenkæder dine oversatte udgydelser med Runtime biblioteker og udformer den endelige HEX fil som kan programmeres ned i din computer.

Filtyper og programmer i programmering fortsat

5. Assembler:

Et pragtfuldt stykke værktøj, som oversætter dit flotte stykke kildetekst arbejde til sproget assembler, maskininstruktioner, et binært sprog, som din maskine direkte kan forstå.-næsten SAMT laver en listing med bl.a. de syntaksfejl assembleren kan finde(IKKE dine logiske fejl)

6. Compiler:

Det værktøj som oversætter dine kildetekstudgydelser i et højniveausprog til noget som linkeren kan bygge videre på. Og listing som assembleren.

7. Listing

En udskrift fra assembler / compiler, som altid viser de programtrin du foreskrev i kildeteksten, samt oftest en hukommelsesplacering og en oversigt. Samt syntaksfejl.

8. Concordance. Et værktøj til at afdække programstrukturen.

9. Disassembler: Oversætter maskinsprog til kildetekst

Program kroppen er altid indeklemt

Mellem to karakteristiske tegn;

I assembler: Som oftest med BEGIN --- END

I sprogene C og C++:

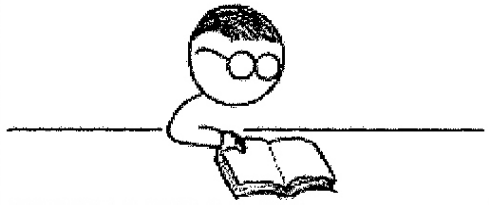
{ (Left opening Brace og (ALT 123)
} (Right closing Brace) og (ALT 125)

Output fra assembler / compiler kan være:

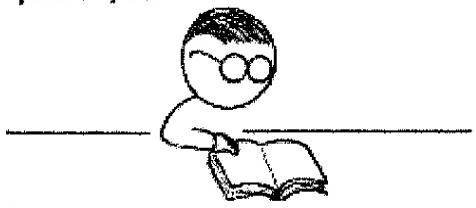
1. Kode som processoren direkte kan forstå, HEX som kan programmeres ned i processoren

2. Kode som skal linkes med et Runtime bibliotek, så de sammen kan danne en eksekverbar fil – HEX – som kan programmeres ned i processoren.

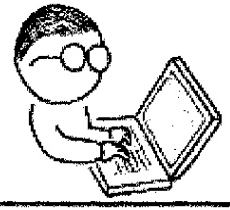
Days 1 - 10
Teach yourself variables, constants, arrays, strings, expressions, statements, functions,...



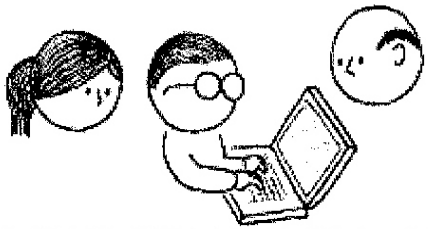
Days 11 - 21
Teach yourself program flow, pointers, references, classes, objects, inheritance, polymorphism, ...



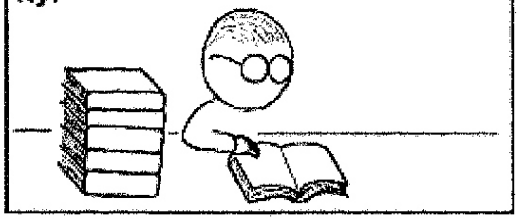
Days 22 - 697
Do a lot of recreational programming. Have fun hacking but remember to learn from your mistakes.



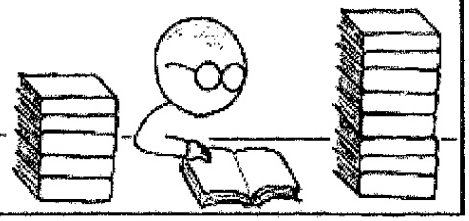
Days 698 - 3648
Interact with other programmers. Work on programming projects together. Learn from them.



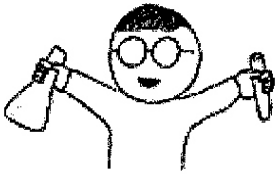
Days 3649 - 7781
Teach yourself advanced theoretical physics and formulate a consistent theory of quantum gravity.



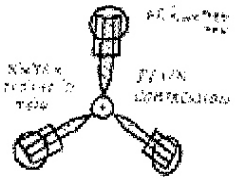
Days 7782 - 14611
Teach yourself biochemistry, molecular biology, genetics,...



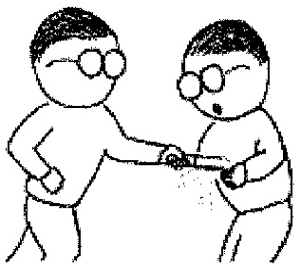
Day 14611
Use knowledge of biology to make an age-reversing potion.



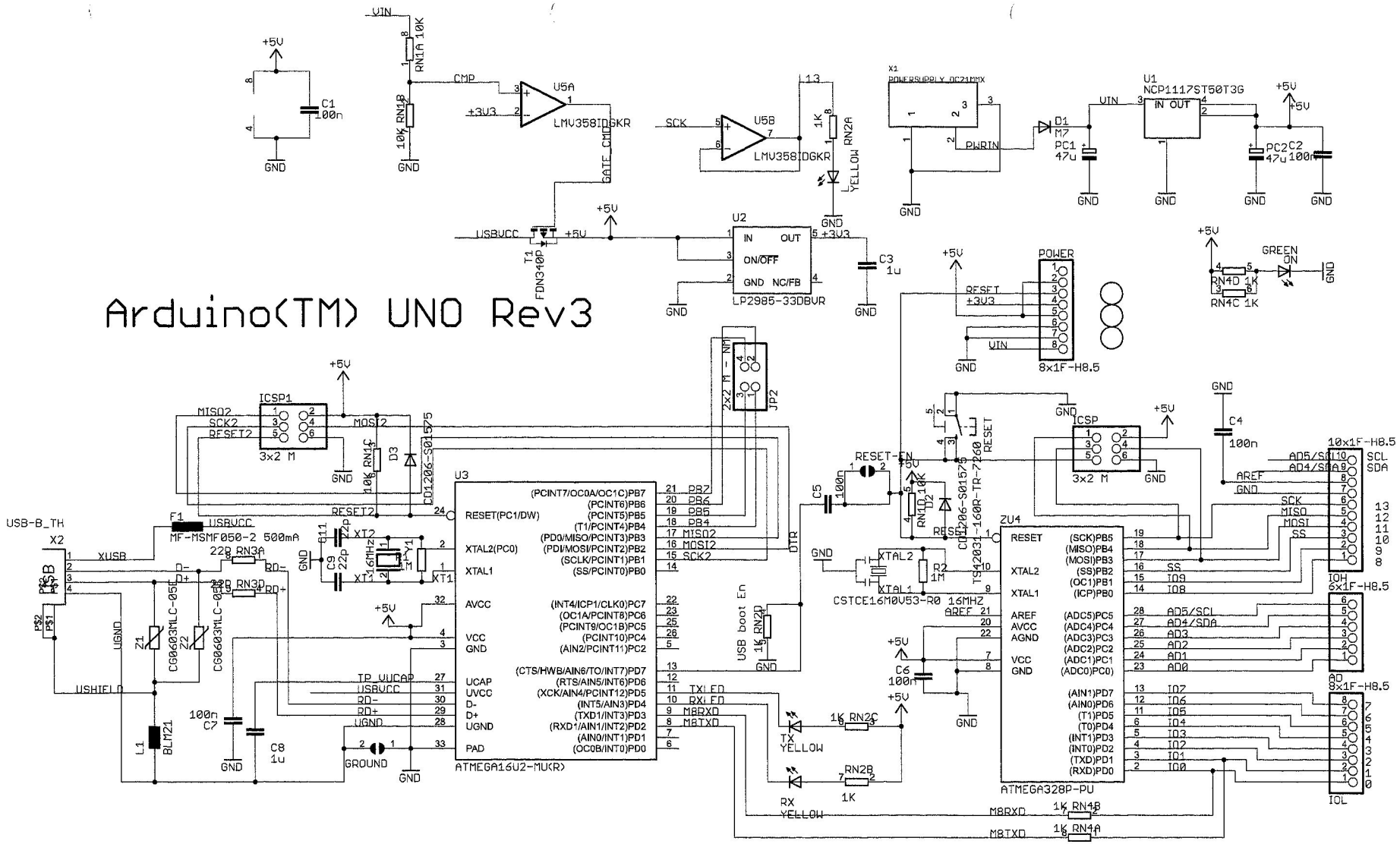
Day 14611
Use knowledge of physics to build flux capacitor and go back in time to day 21.



Day 21
Replace younger self.



Hvad en PIC kan indeholde



Hvad en processor kan indeholde

1. Flash memory, indeholder DIT eksekverbare program. 10.000

2. RAM: Hukommelse, hvor processoren kan skrive i
   og læse fra. Nå man tænder er indholdet uforudsigeligt.

3. EEPROM. Lager hvor processoren kan skrive og læse, MEN det den sidst skrev i en EEprom adresse er uændret når man tænder igen.

4. Registre: Programregisteret er det vigtigste register.
   Det indeholder altid adressen på den næste instruktion.

   Desuden findes der flere dedikerede registre, nogle til specielle anvendelser, som f.eks. Bitkodning af tilladelser,,,,

   Eksempel: Interrupt enable, som hvis kodet sand tillader at en eller flere af interrupt mulighederne kan fungere.

Hvad en processor kan indeholde
fortsat

5. En akkumulator. Det register der ALTID er med i alle regneoperation, og hvor resultatet altid ender op. Kan hedde A register eller W register eller noget helt andet.

6. Diverse hjælperegistre, f.eks. Index register, som anvendes når man skal pege på en bestemt plads i en række af f.eks. Karakterer.

Index = pegepind.

Indhold fortsat

7. Input og output ben,
   benævnt porte/gates. Næsten alle kan under programkontrol sættes til at være input eller output med eller uden pullup

   Mange andre muligheder forefindes, de skal ALLE programmeres i den del af programmet man oftest kalder INIT.

   Pullup: En indbygget modstand op til +5volt forsyningsspænding

8.Serielle in og output ben.
   Kaldes oftest forkert for RS232, men bør nærmere være RS422/425 ?
   Ifølge RS232 normen skal man køre med et sving på +/- 15 volt, det kan vi jo ikke når vi kun har 5 volt.

   Hastighed i Baud, rundt regnet bits/sekund.

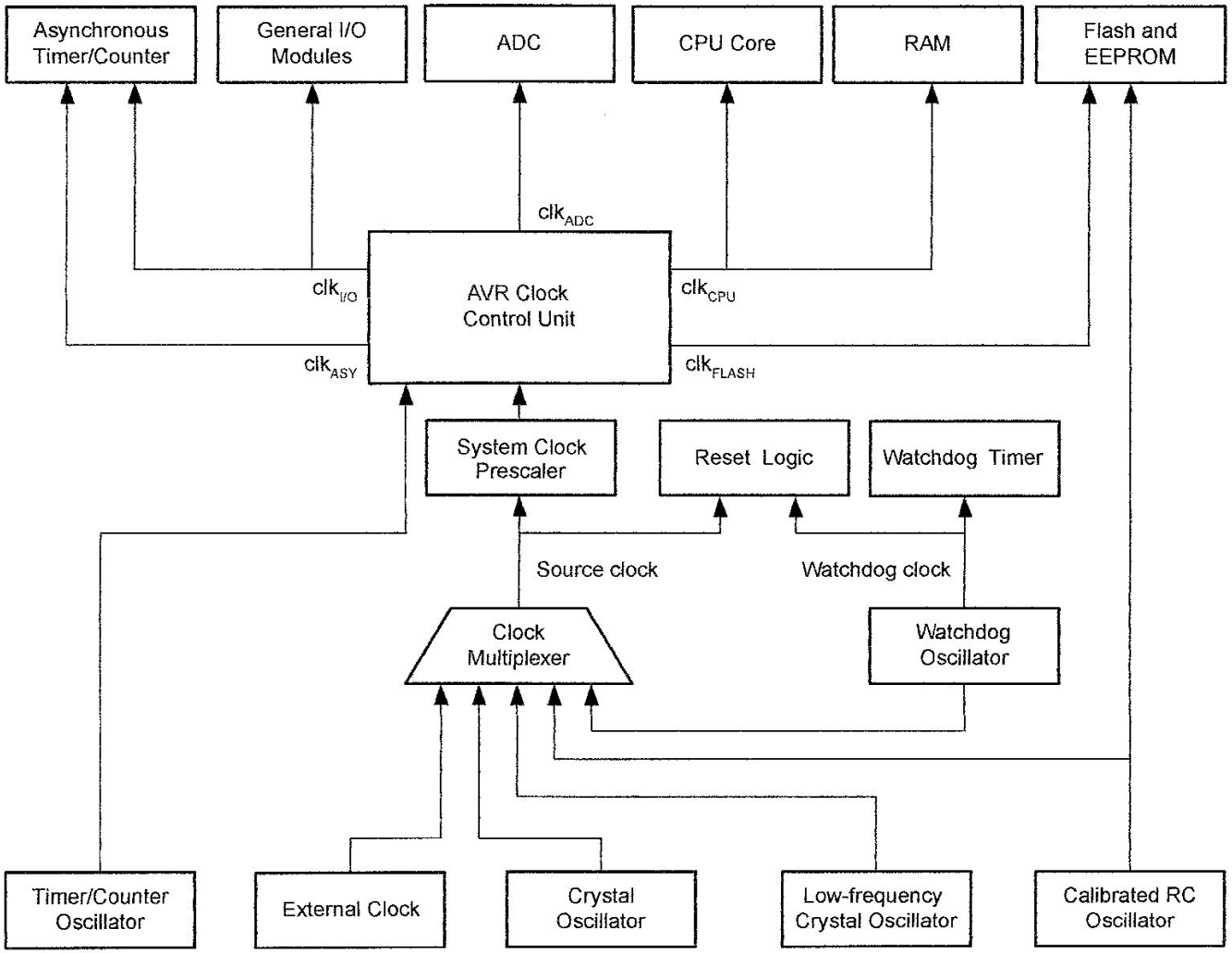
Indhold fort-fortsat

9. A/D konverter input.

10. A/D konverter
    spændingsforsyning og stel uafhængig af resten af chippen.
    Processoren (+ I/O clock) kan lulles i søvn mens man
    A/D konverterer for at reducere støjgener.

11. 2 ben til ydre krystal.
    Processoren har mindst 4 forskellige muligheder for oscillatorvalg,
    herunder en indbygget og fabrikskalibreret oscillator.

Figure 8-1. Clock Distribution



032		033	!	034	"	035	#	036	$	037	%	038	&	039	'
040	(	041	)	042	*	043	+	044	,	045	-	046	.	047	/
048	0	049	1	050	2	051	3	052	4	053	5	054	6	055	7
056	8	057	9	058	:	059	;	060	<	061		062	>	063	?
064	@	065	A	066	B	067	C	068	D	069	E	070	F	071	G
072	H	073	I	074	J	075	K	076	L	077	M	078	N	079	O
080	P	081	Q	082	R	083	S	084	T	085	U	086	V	087	W
088	X	089	Y	090	Z	091	[	092	\	093	]	094	^	095	_
096	`	097	a	098	b	099	c	0100	d	0101	e	0102	f	0103	g
0104	h	0105	i	0106	j	0107	k	0108	l	0109	m	0110	n	0111	o
0112	p	0113	q	0114	r	0115	s	0116	t	0117	u	0118	v	0119	w
0120	x	0121	y	0122	z	0123	{	0124	|	0125	}	0126	~	0127	
0128	€	0129		0130	‚	0131	ƒ	0132	„	0133	…	0134	†	0135	‡
0136	ˆ	0137	‰	0138	Š	0139	‹	0140	Œ	0141		0142		0143	
0144		0145	‘	0146	’	0147	“	0148	”	0149	•	0150	–	0151	—
0152	˜	0153	™	0154	š	0155	›	0156	œ	0157		0158		0159	Ÿ
0160		0161	¡	0162	¢	0163	£	0164	¤	0165	¥	0166	¦	0167	§
0168	¨	0169	©	0170	ª	0171	«	0172	¬	0173	-	0174	®	0175	¯
0176	°	0177	±	0178	2	0179	3	0180	´	0181	µ	0182	¶	0183	·
0184	¸	0185	1	0186	º	0187	»	0188	¼	0189	½	0190	¾	0191	¿
0192	À	0193	Á	0194	Â	0195	Ã	0196	Ä	0197	Å	0198	Æ	0199	Ç
0200	È	0201	É	0202	Ê	0203	Ë	0204	Ì	0205	Í	0206	Î	0207	Ï
0208	Ð	0209	Ñ	0210	Ò	0211	Ó	0212	Ô	0213	Õ	0214	Ö	0215	×
0216	Ø	0217	Ù	0218	Ú	0219	Û	0220	Ü	0221	Ý	0222	Þ	0223	ß
0224	à	0225	á	0226	â	0227	ã	0228	ä	0229	å	0230	æ	0231	ç
0232	è	0233	é	0234	ê	0235	ë	0236	ì	0237	í	0238	î	0239	ï
0240	ð	0241	ñ	0242	ò	0243	ó	0244	ô	0245	õ	0246	ö	0247	÷
0248	ø	0249	ù	0250	ú	0251	û	0252	ü	0253	ý	0254	þ	0255	ÿ

Mange års erfaringer!!!!!

1. SKRIV KOMMENTARER –
   Fortæl hvad du vil lave med dette.

2. SKRIV KOMMENTARER -- **TIL ALT**

3. Skriv kommentarer,
   fortæl til dine beundrere hvilke genistreger du her udfører.

4. Skriv kommentarer.

TYPISK:

Når du skifter din akkumulator 1 bit til venstre

SÅ SKRIVER DU IKKE: Jeg ganger med 2 -- ude i kommentaren

DU SKRIVER HVORFOR du ganger med 2

5. Brug menneskelige definitioner (EQATES.)

Eksempel:

{

INIT: ottehex equ Byte }

dikkaffe: ….. Her defineres Subroutine drikkaffe

……

Return

Og så skal den bruges:

{

Progra: …..

…

ottehex = 00H Nulstiller min hex tæller

Repeat

ottehex = ottehex plus én

GOSUB drikkaffe

until

ottehex = 08H igennem loopen 8 gange

}

Han fik otte kopper kaffe! SYNTAKSFEJL

Det du ikke skal gøre

Lav et program som beregner hvordan en instruktion længere fremme i forløbet skal være. (KUN for assemblerfreaks)

```
        instr.      Beregn del af instruktion til bombe
        instr.      Og den næste del
        instr,      og den tredje del
        Gem resultat under navnet bombe
        instr.
Bombe:  HA HA, DU kan bare ikke regne ud hvad den gør.!!!!
        instr,
        …. o.s.v.
```

Eller endnu bedre ide

.

,

IF

Erik er væk fra klubbernes medlemsliste

THEN

slet alle programmer på harddiskene og crash
alle Hillerød og Birkerød afdelingernes computere

ELSE
continue

.

.

Sådan laver du et program.

Med tekst editor: Skriv dine genistreger i en kildetekst
(EDITOR = IKKE tekstbehandler)

Som hældes ned i en Compiler (Assembler),
Som også har fat i et KILDETEKST LIBRARY

Ud kommer en ufærdig fil

Som opfanges af en linker
SOM OGSÅ har fat i RUNTIME LIBRARY-

Ud kommer en HEX fil.

som du med processorens ”Boot Loader” downloader ned i din Arduino Uno board's flash memory

Og så gør du som de små hunde: PRØVER.

(Når du bliver skrap har vi også en debugger)

Programstrukturer der går igen i alle sprog

IF

(Noget er sandt, et boolsk udtryk)

THEN

Gør noget

ELSE

Gør noget andet.

Et eksempel:
IF
(solen skinner)
THEN
Sov i solen
ELSE
Sov i sengen.

WHILE

(Et eller andet er sandt, boolsk igen)

DO

Udfør jobbet

Et eksempel:

WHILE
Der er flere pilsnere i kassen
DO
Knap bajere op.

BEMÆRK: Her tester vi tilstanden før vi udfører jobbet

REPEAT
Udfør jobbet
UNTIL
Noget er sandt, boolsk udtryk

REPEAT
Drik bajere
UNTIL
Du vælter

Her udfører vi jobbet før vi tester.

ADVARSEL

I alle loops, IF, REPEAT, WHILE

Er der en fare for at afslutningskriteriet IKKE bliver opfyldt.

REPEAT
Læs karakterer fra input
UNTIL
Char = 714

Hvis 714 ALDRIG optræder bliver vore computer hængende her så længe der er strøm på.

Kan klares med en tæller:

Tæller = 1000
REPEAT
Læs karakterer fra input
Tæller = tæller - 1
UNTIL
(Char = 714) OR (Tæller = 0)

Eller en anden løsning

I INIT indsættes:

Watchdog enable (og lidt mere)

I programmet indsættes nu på strategiske steder:
RESET Watchdog tæller

Hvis Watchdog tæller ikke er resat og der er gået 38 milisekunder (typisk værdi som programmeres i INIT), så aktiverer Watchdog en interrupt, som så kan udføre en passende aktion.

Interrupt: Programmet afbrydes og fortsætter programeksekveringen fra en specifik adresse,
glæd dig til senere.

Almindelige programmer starter altid i 00hex
Glæd dig til senere.

Subrutiner

Alle sprog har en form for et subrutine kald, oftest med ordren CALL.
Subrutiner bruges når de samme programtrin skal udføres fra forskellige steder i programmet -- eller for at strukturere et program.

```
BEGIN           Programkroppen
Ordre 36
Ordre 37
CALL FINDOST
….
Call FINDOST
.
END

FINDOST         Her er subrutinen
Ordre
Ordre
RETURN
```

Returadressen - når den skal retur fra FINDOST - POPPES fra stacken.
Den blev pushet ved CALL.

Function

Er en begavet form for subrutine, som udfører typisk en regne operation og returnerer en værdi.

Stack

Stack en en buffer af type LIFO, Last In first out = et hul i væggen)

Den bruges primært når en subrutine skal aktiveres med et CALL statement.
Ved CALL pusher man RETUR adressen på stacken.
Når subrutine afsluttes (med statementet RETURN) så popper processoren sin returadresse fra stacken.

Principielt kan du pushe og poppe alt hvad du vil, MEN sædvanligvis har du kun en dybde på 8 x 16 bit.

Stacken ligger i 328p processoren i toppen af RAM' en.
Stack pointer skal initialiseres når du arbejder i assembler.

De gode gamle dage !

Du tog et programmeringsprint,

fodrede det fra en power supply
forbandt det til din PC
isatte din processor
Programmerede processoren
Flyttede den til sit endelige sted i
et andet nyt print
Prøvekørte

NYE TIDER.

Med Arduino Uno

USB kabel mellem PC og Arduino Uno
løs power supply fra skuffen
programmerer processoren og prøvekører.

Det kaldes ”Boot Loader” programmering,

Den programmeres på det sted
hvor den skal blive
indtil den skal på genbrugspladsen.

Og så koster en Arduino Uno (Print og hele 2 processorer)

Kun 199 kr. OG er version 3, sidste nye skrig.

Det dertil egnede display koster 99 kr,
Og så skal du bruge en USB minikabel.
Softwaren downloader du GRATIS fra Arduino.

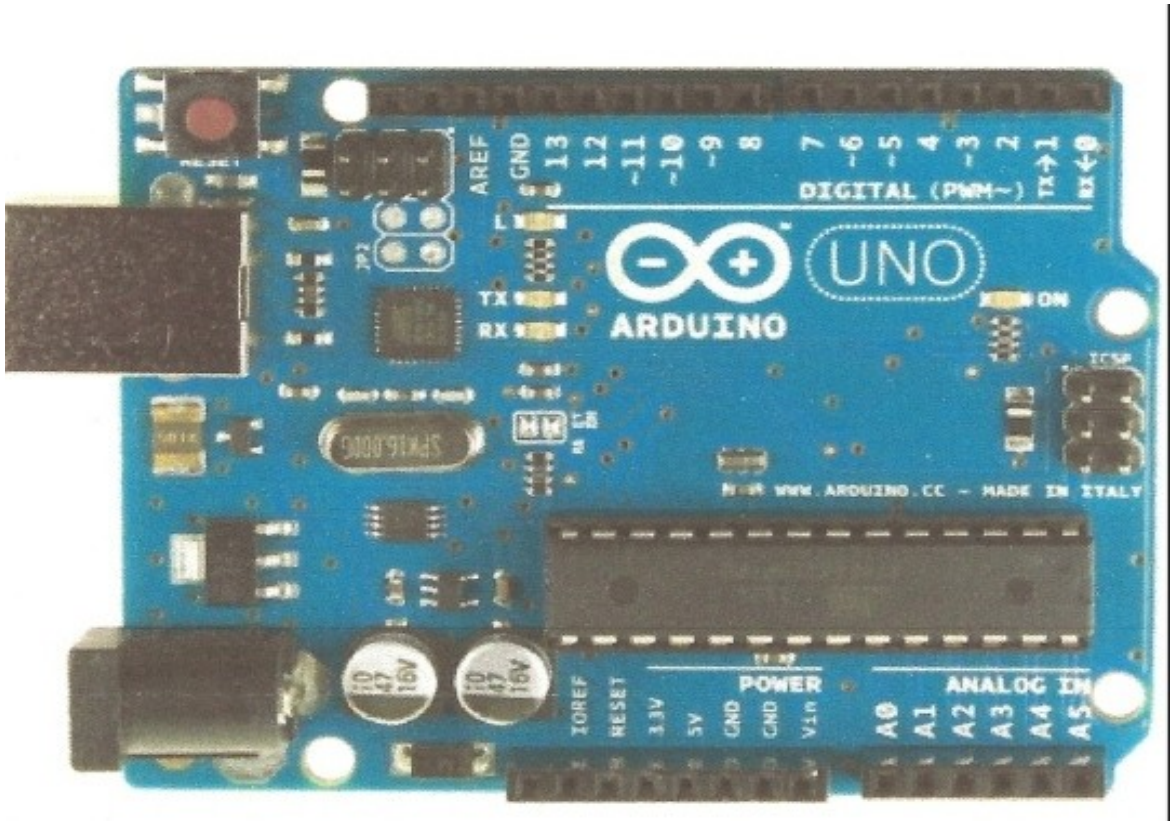
OG VIGTIGST: Du kan arbejde seriøst hjemme ved din egen PC.

Derfor **ikke mere ”Gode Gamle Dage”**

OG så er den anvendte processor Atmel ATmega328P
”TOP” typen = DEN STØRSTE

Instruktionsmanualen er på 464 sider = 3 ringbind.

Manualen til Arduino Uno board er kun 1 ringbind.



Sådan ser vores fremtidige processor board ud

Egenskaber

Power fra USB eller direkte, automatisk valg

Voltin: 6-20 volt, anbefalet dog 7-20 volt

(Hen i skuffen og find en 220V/12 Vdc af de sorte)

Flash prom: 32 kbyte (Plads til C program, det fylder meget)

In/Out porte
seriel in/out (USART),
2 externe interrupts,
8 bit PWM output,
6 analoge input,
2 stk 10 bit A/D,
Extern referencespænding for A/D
II Wire
Og meget -- meget mere.

Støvlestrop = Bootstrap

Sådan ser en bootstrap til en Varian 620-L100 ud.
Octal selvfølgelig. Den indlæser en ”Binær Loader”,
Som derefter kan bruges til at indlæse programmer.
NB: Her læser vi hulstrimmel.

Table 12-1. Bootstrap Loader Routines

Address	High-Speed Reader Code	Teletype Reader Code	Symbolic Coding
007756	102637	102601	READ CIB RDR
007757	004011	004011	ASLB NBIT – 7
007760	004041	004041	LRLB 1
007761	004446	004446	LLRL 6
007762	001020	001020	JBZ SEL
007763	007772	007772	(Memory address)
007764	055000	055000	STA 0,1
007765	001010	001010	JAZ LHLT + 1
007766	007000*	007000*	(Memory address)
007767	005144	005144	IXR
007770	005101	005101	ENTR INCR 1
007771	100537	102601	EXC** RDON
007772	101537	101201	SEL SEN IBFR,READ
007773	007756	007756	(Memory address)
007774	001000	001000	JMP * – 2
007775	007772	007772	Memory address)

NOTE

The bootstrap loader routine is always loaded into the highest address of the first 4K memory

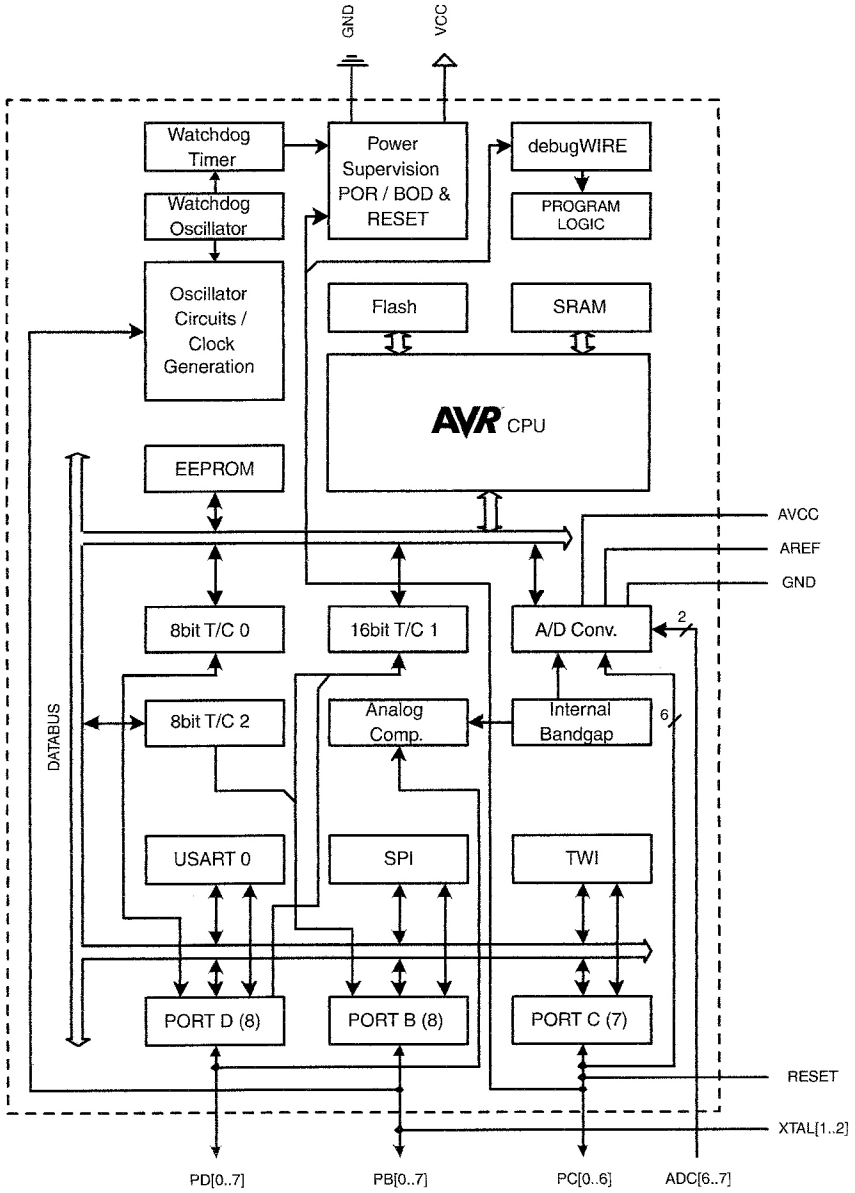
**CIB Instruction if Teletype bootstrap

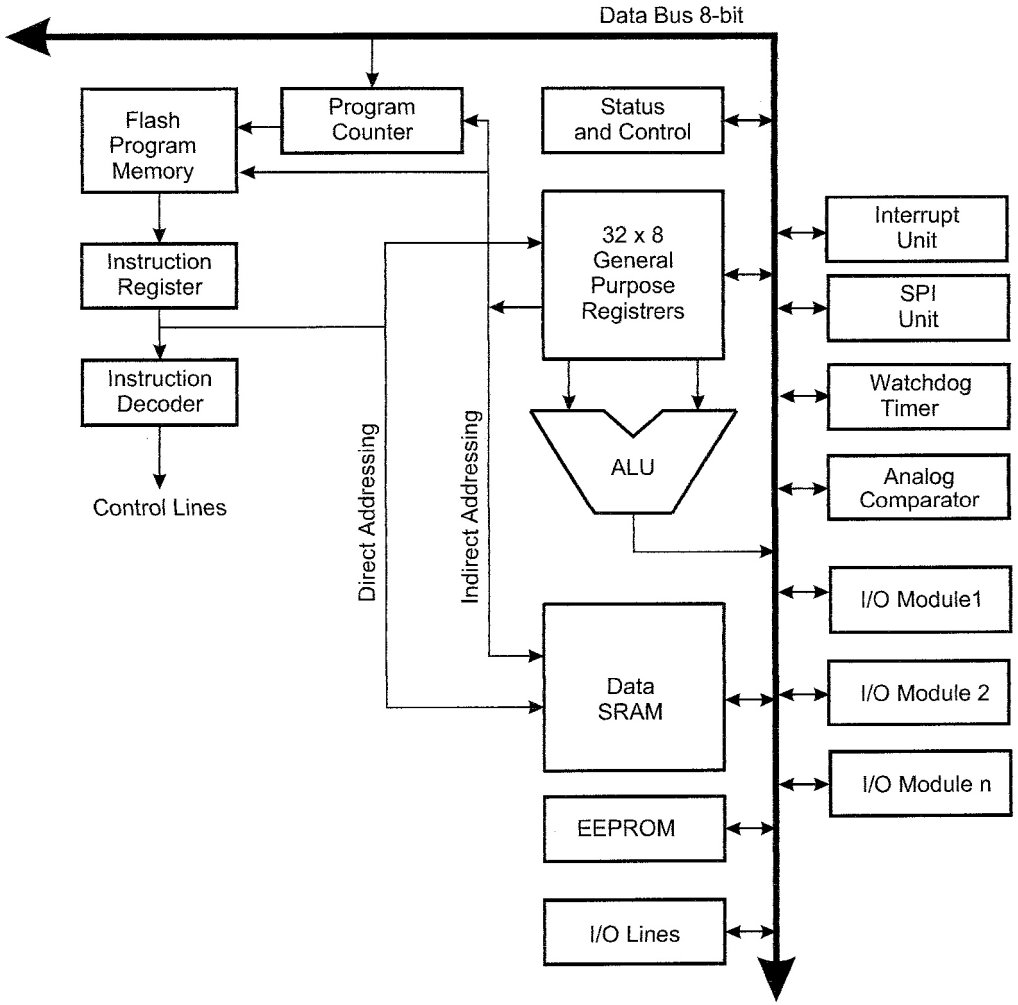
Atmel ATmega328P har principielt 32 k flash memory,

Men det passer ikke helt.

1 pund --- nå nej et halvt kilo er reserveret til en bootloader.

Den er vores gode ven når vi skal fylde et program
hen i flash memoryen.


GND
VCC
Watchdog Timer
Watchdog Oscillator
Power Supervision POR / BOD & RESET
debugWIRE
PROGRAM LOGIC
Oscillator Circuits / Clock Generation
Flash
SRAM
AVR CPU
EEPROM
AVCC
AREF
GND
8bit T/C 0
16bit T/C 1
A/D Conv.
2
DATABUS
8bit T/C 2
Analog Comp.
Internal Bandgap
6
USART 0
SPI
TWI
PORT D (8)
PORT B (8)
PORT C (7)
RESET
XTAL[1..2]
PD[0..7]
PB[0..7]
PC[0..6]
ADC[6..7]


Data Bus 8-bit
Flash Program Memory
Program Counter
Status and Control
Instruction Register
Instruction Decoder
Control Lines
32 x 8 General Purpose Registrers
ALU
Direct Addressing
Indirect Addressing
Interrupt Unit
SPI Unit
Watchdog Timer
Analog Comparator
I/O Module1
I/O Module 2
I/O Module n
Data SRAM
EEPROM
I/O Lines

The most typical and general program setup for the Reset and Interrupt Vector Addresses in ATmega328P is:

```
Address  Labels Code                 Comments
0x0000           jmp    RESET        ; Reset Handler
0x0002           jmp    EXT_INT0     ; IRQ0 Handler
0x0004           jmp    EXT_INT1     ; IRQ1 Handler
0x0006           jmp    PCINT0       ; PCINT0 Handler
0x0008           jmp    PCINT1       ; PCINT1 Handler
0x000A           jmp    PCINT2       ; PCINT2 Handler
0x000C           jmp    WDT          ; Watchdog Timer Handler
0x000E           jmp    TIM2_COMPA   ; Timer2 Compare A Handler
0x0010           jmp    TIM2_COMPB   ; Timer2 Compare B Handler
0x0012           jmp    TIM2_OVF     ; Timer2 Overflow Handler
0x0014           jmp    TIM1_CAPT    ; Timer1 Capture Handler
0x0016           jmp    TIM1_COMPA   ; Timer1 Compare A Handler
0x0018           jmp    TIM1_COMPB   ; Timer1 Compare B Handler
0x001A           jmp    TIM1_OVF     ; Timer1 Overflow Handler
0x001C           jmp    TIM0_COMPA   ; Timer0 Compare A Handler
0x001E           jmp    TIM0_COMPB   ; Timer0 Compare B Handler
```

```
0x0020          jmp     TIM0_OVF        ; Timer0 Overflow Handler
0x0022          jmp     SPI_STC         ; SPI Transfer Complete Handler
0x0024          jmp     USART_RXC       ; USART, RX Complete Handler
0x0026          jmp     USART_UDRE      ; USART, UDR Empty Handler
0x0028          jmp     USART_TXC       ; USART, TX Complete Handler
0x002A          jmp     ADC             ; ADC Conversion Complete Handler
0x002C          jmp     EE_RDY          ; EEPROM Ready Handler
0x002E          jmp     ANA_COMP        ; Analog Comparator Handler
0x0030          jmp     TWI             ; 2-wire Serial Interface Handler
0x0032          jmp     SPM_RDY         ; Store Program Memory Ready Handler
;
0x0033RESET:    ldi     r16, high(RAMEND); Main program start
0x0034          out     SPH,r16         ; Set Stack Pointer to top of RAM
0x0035          ldi     r16, low(RAMEND)
0x0036          out     SPL,r16
0x0037          sei                     ; Enable interrupts
0x0038          <instr>  xxx
  ...     ...     ...  ...
```

Efterårets mulige emner

1. Power Meter.

   At måle hvor meget power der afsættes i en indbygget 50 ohms afslutnings modstand.
   Det bliver en dybtgående demonstration af sproget C
   og dets samspil med ATmega328p processoren
   Måske med ekstraprint med blandt andet en IC ved navn 8307.
   Måske får vi print til projektet fra OZ9QV til 8307 med mere.
   Målsætning: 100W, 10 – 500 Mhz, hvis vi kan køle modstanden.

2. Pilot tone generator.

   Vi, der kører med IC 910, mangler pilottoner til repeaterne.
   Dem laver vi på en Arduino.

3. Elektronisk kompas.

   Kører du moonbounce eller tracker på satellitter henter og viser vi kompasretning, tilt og roll, samt temperatur på din antenne.
   Selve kompasset henter vi hos Robotfolket, vores opgave er at hente dets data og præsenterer dem på vores display.