Run-time environment and program library

Ur innehållet:

Olika typer av bibliotek

- Kompilatorbibliotek
- C-bibliotek

Run-time miljö

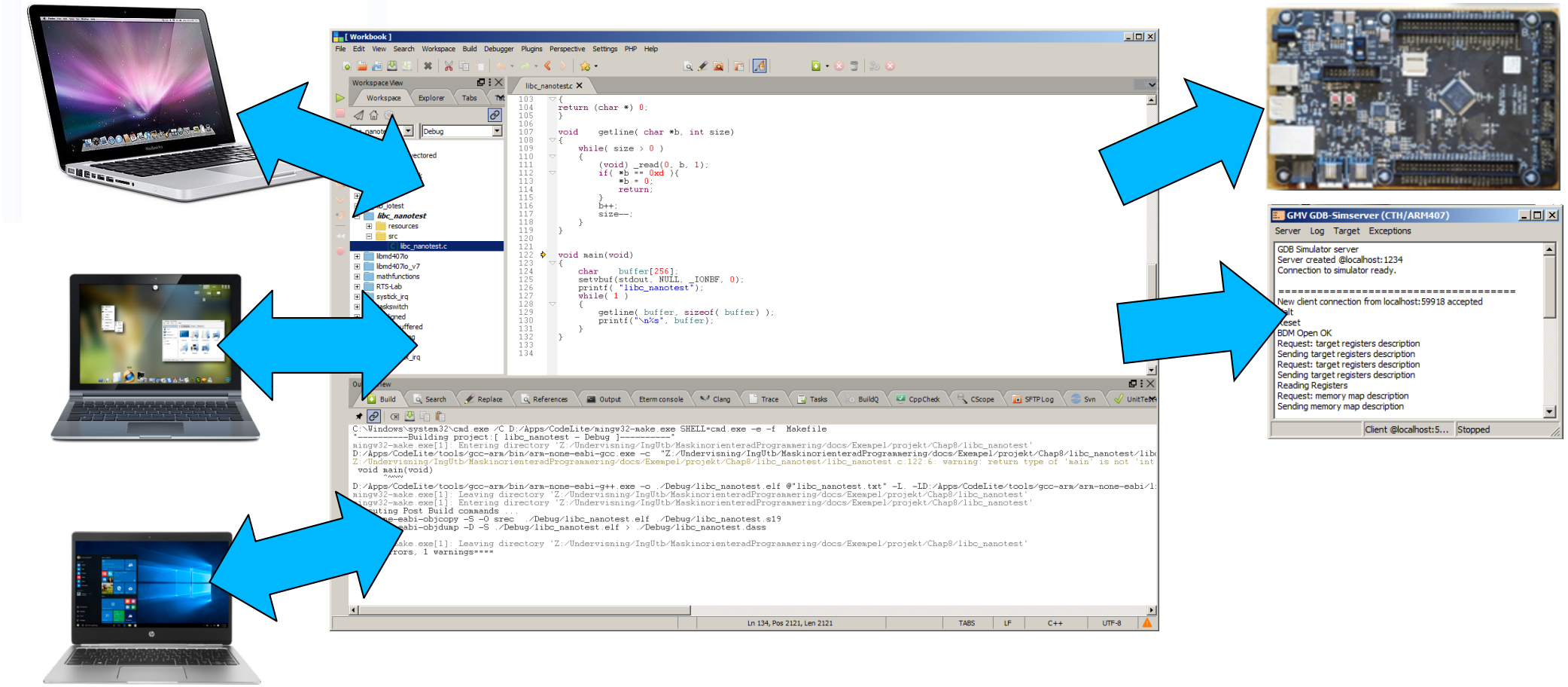
Så skapar du ett nytt bibliotek

Läsanvisningar:

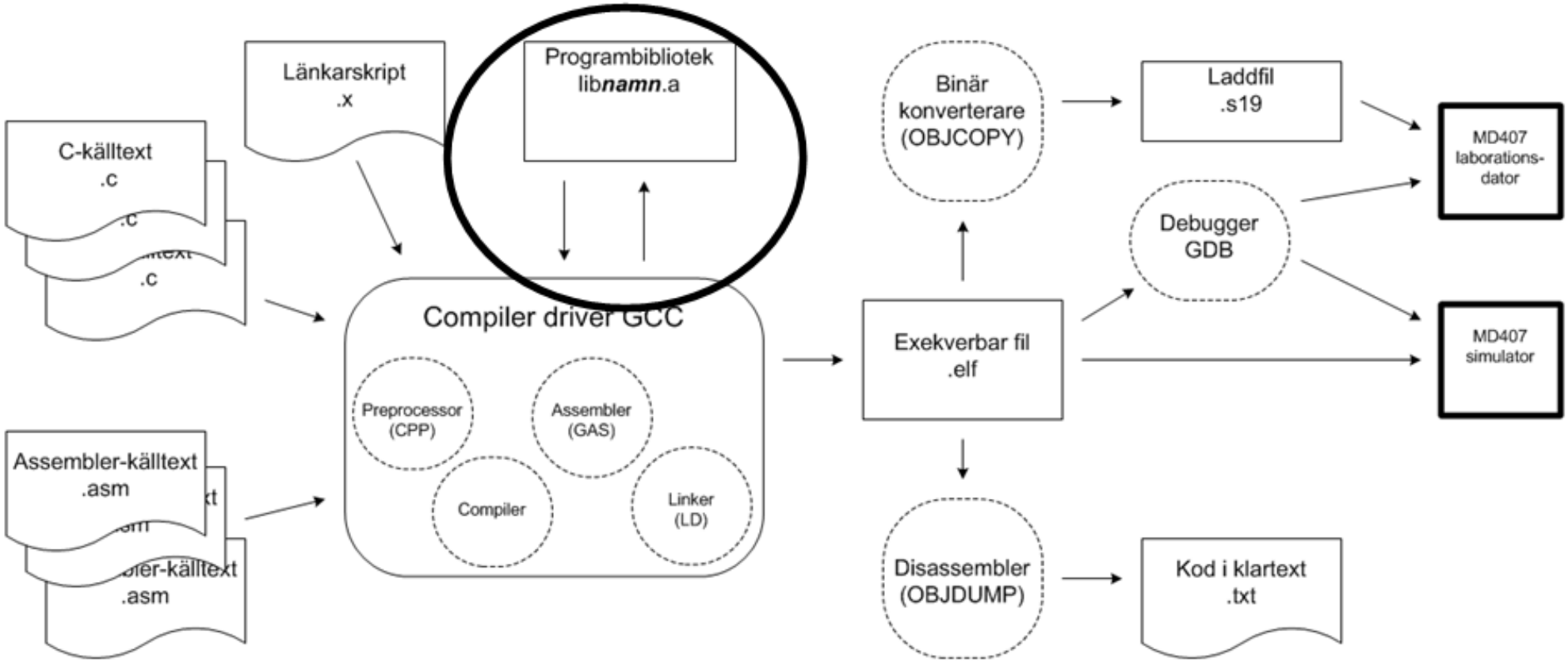
Arbetsbok kapitel 8

Host computer and target computer:

Konventionell programutveckling sker *på värddator för värddator*.
Korsutveckling sker *på värdator för måldator*, i vårt fall MD407.



Program Library



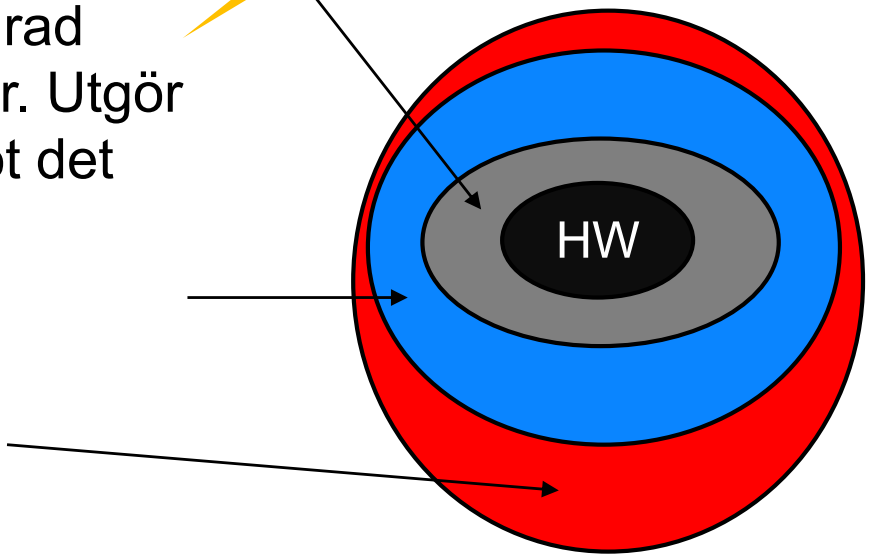
Program library – different types

Programbibliotek kan ofta ses som "kompatibilitetslager" där en anpassning sker till någon speciell maskinvara och något specifikt operativsystem.

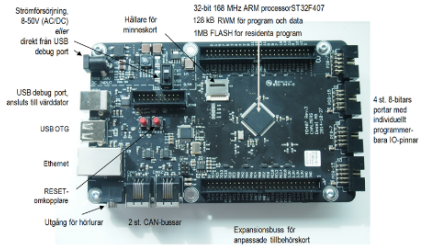
- Kompilatorbibliotek – tillhandahåller det stöd som krävs för att ett standardiserat C-program korrekt ska kunna översättas till maskinkod, oavsett processorarkitektur.
- Standard C-bibliotek – tillhandahåller en lång rad funktioner användbara i de flesta applikationer. Utgör också vanligtvis det viktigaste gränssnittet mot det använda operativsystemet.
- Användarspecifika bibliotek.

FP operations?

printf…



GUI: Windows and menus?

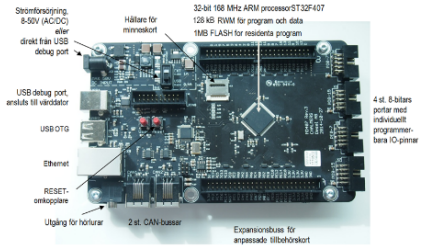


Target computer – architecture

En måldator kan karakteriseras av:
arkitektur, *minnesdisposition* och *periferienheter*.

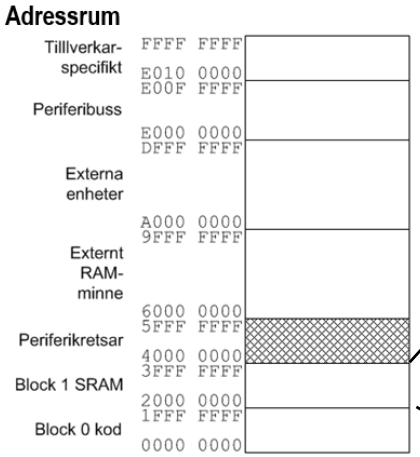
- Arkitektur – ARM-Cortex, tillhandahåller processorer som hanterar olika typer av instruktionsuppsättningar:
  - Thumb 1
  - Thumb 2
  - DSP (*Digital Signal Processing*)
  - FP (*Floating Point*)

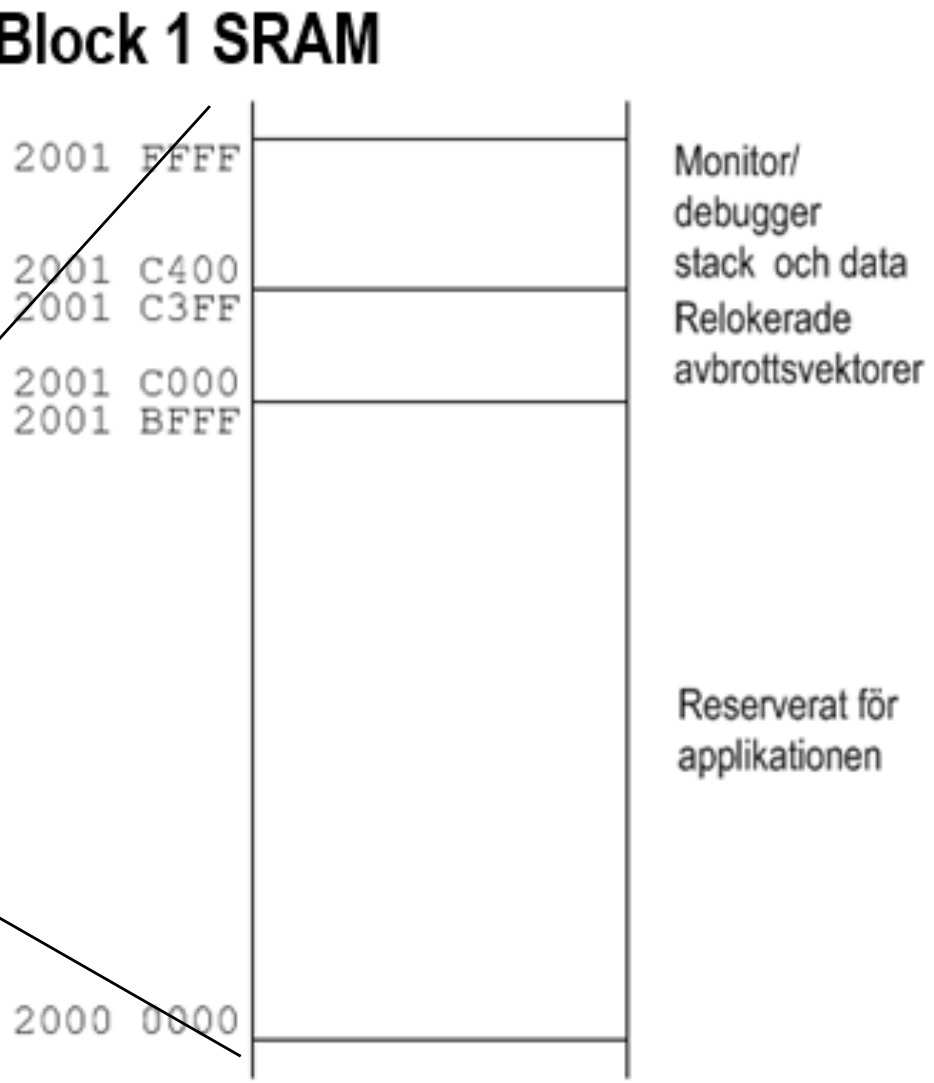
Instruktion	Storlek	Cortex M0	Cortex M0+	Cortex M1	Cortex M3	Cortex M4	Cortex M7	Ark.
ADC, ADD, (ADR), AND, ASR, B, BIC, BKPT, BLX, BX, CMN, CMP, CPS, EOR, LDM, (LDMIA, LDMFD), LDR, LDRB, LDRH, LDRSB, LDRSH, LSL, LSR, MOV, MUL, MVN, NOP, ORR, POP, PUSH, REV, REV16, REVSH, ROR, RSB, SBC, SEV, STM, (STMIA, STMEA), STR, STRB, STRH, SUB, SVC, SXTB, SXTH, TST, UXTB, UXTH, WFE, WFI, YIELD	16-bit	x	x	x	x	x	x	v6
BL, DMB, DSB, ISB, MRS, MSR	32-bit	x	x	x	x	x	x	
CBNZ, CBZ, IT	16-bit				x	x	x	
ADC, ADD, AND, ASR, B, BFC, BFI, BIC, CDP, CLREX, CLZ, CMN, CMP, DBG, EOR, LDC, LDMA, LDMDB, LDR, LDRB, LDRBT, LDRD, LDREX, LDREXB, LDREXH, LDRH, LDRHT, LDRSB, LDRSBT, LDRSHT, LDRT, MCR, LSL, LSR, MLS, MCRR, MLA, MOV, MOVT, MRC, MRRC, MUL, MVN, NOP, ORN, ORR, PLD, PLDW, PLI, POP, PUSH, RBIT, REV, REV16, REVSH, ROR, RRX, RSB, SBC, SBFX, SDIV, SEV, SMLAL, SMULL, SSAT, STC, STMDB, STR, STRB, STRBT, STRD, STREX, STREXB, STREXH, STRH, STRHT, STRT, SUB, SXTB, SXTB, TBB, TBH, TEQ, TST, UBFX, UDIV, UMLAL, UMULL, USAT, UXTB, UXTH, WFE, WFI,YIELD	32-bit				x	x	x	v7
PKH, QADD, QADD16, QADD8, QASX, QDADD, QDSUB, QSAX, QSUB, QSUB16, QSUB8, SADD16, SADD8, SASX, SEL SHADD16, SHADD8, SHASX, SHSAX, SHSUB16, SHSUB8, SMLABB, SMLABT, SMLATB,SMLATT, SMLAD, SMLALBB, SMLALBT, SMLALTB, SMLALTTSMLALD, SMLAWB, SMLAWT, SMLSD, SMLSLD, SMMLA, SMMLS, SMMUL, SMUAD, SMULBB, SMULBT, SMULTT, SMULTB, SMULWT, SMULWB, SMUSD, SSAT16, SSAX, SSUB16, SSUB8, SXTAB, SXTAB16, SXTAH, SXTB16, UADD16, UADD8, UASX, UHADD16, UHADD8, UHASX, UHSAX, UHSUB16, UHSUB8, UMAAL, UQADD16, UQADD8, UQASX, UQSAX, UQSUB16, UQSUB8, USAD8, USADA8, USAT16, USAX, USUB16, USUB8, UXTAB, UXTAB16, UXTAH, UXTB16	32-bit					x	x	v7e (DSP)
VABS, VADD, VCMP, VCMPE, VCVT, VCVTR, VDIV, VLDM, VLDR, VMLA, VMLS, VMOV, VMRS, VMSR, VMUL, VNEG, VNMLA, VNMLS, VNMUL, VPOP, VPUSH, VSQRT, VSTM, VSTR, VSUB	32-bit					SP FPU	SP FPU	32-b FP
FP- dubbel precision	32-bit						DP FPU	64-b FP



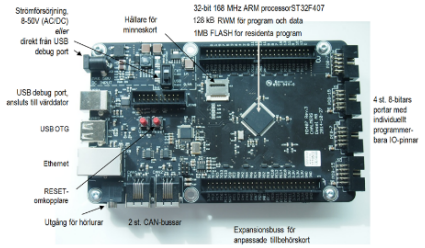
Target computer – memory map

- Minneskonfiguration – Hur mycket minne finns det och var är det placerat i adressrummet?





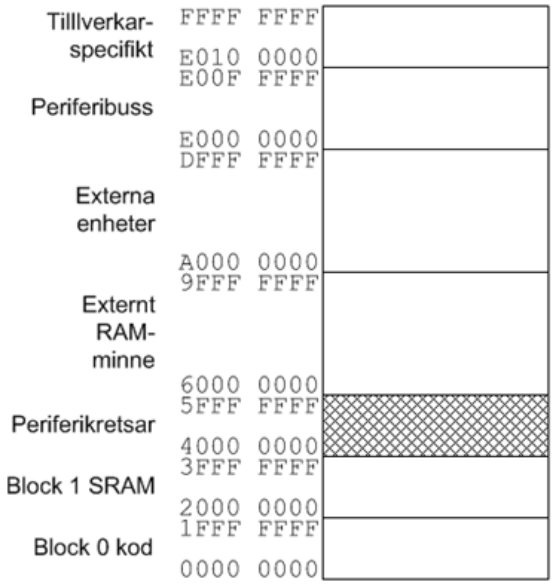
Informationen måste finnas för att vi exempelvis ska kunna implementera dynamisk minneshantering med ”malloc/free”



Target computer – peripherals

- Tillgängliga periferienheter –
  - Systemenheter
  - IO-enheter

Adressrum



Periferibuss

E000 EF44	"Floating point unit"
E000 EF30	"NVIC"
E000 EF03	"Memory protection unit"
E000 EF00	"Floating point unit, access control"
E000 EDB8	"System Control Block"
E000 ED90	"NVIC"
E000 ED8B	"System timer"
E000 ED88	
E000 ED3F	
E000 ED00	
E000 E4EF	
E000 E100	
E000 E01F	
E000 E010	

Periferikretsar

A000 0000 - A000 0FFF	FSMC control reg	AHB3
5006 0800 - 5006 0BFF	RNG	AHB1
5006 0400 - 5006 07FF	HASH	
5006 0000 - 5006 03FF	CRYP	
5005 0000 - 5005 03FF	DCMI	
5000 0000 - 5003 FFFF	USB OTG FS	
4004 0000 - 4007 FFFF	USB OTG HS	
4002 B000 - 4002 BBFF	DMA2D	
4002 9000 - 4002 93FF	ETHERNET MAC	
4002 8C00 - 4002 8FFF		
4002 8800 - 4002 8BFF		
4002 8400 - 4002 87FF		
4002 8000 - 4002 83FF		
4002 6400 - 4002 67FF	DMA2	
4002 6000 - 4002 63FF	DMA1	
4002 4000 - 4002 4FFF	BKPSRAM	
4002 3C00 - 4002 3FFF	Flash interface	
4002 3800 - 4002 3BFF	RCC	
4002 3000 - 4002 33FF	CRC	
4002 2800 - 4002 2BFF	GPIOK	
4002 2400 - 4002 27FF	GPIOJ	
4002 2000 - 4002 23FF	GPIOI	
4002 1C00 - 4002 1FFF	GPIOH	
4002 1800 - 4002 1BFF	GPIOG	
4002 1400 - 4002 17FF	GPIOF	
4002 1000 - 4002 13FF	GPIOE	
4002 0C00 - 4002 0FFF	GPIOD	
4002 0800 - 4002 0BFF	GPIOC	
4002 0400 - 4002 07FF	GPIOB	
4002 0000 - 4002 03FF	GPIOA	
4001 6800 - 4001 6BFF	LCD-TFT	APB2
4001 5800 - 4001 5BFF	SAI1	
4001 5400 - 4001 57FF	SPI6	
4001 5000 - 4001 53FF	SPI5	
4001 4800 - 4001 4BFF	TIM11	
4001 4400 - 4001 47FF	TIM10	
4001 4000 - 4001 43FF	TIM9	
4001 3C00 - 4001 3FFF	EXTI	
4001 3800 - 4001 3BFF	SYSCFG	
4001 3400 - 4001 37FF	SPI4	
4001 3000 - 4001 33FF	SPI1	
4001 2C00 - 4001 2FFF	SDIO	
4001 2000 - 4001 23FF	ADC1-ADC2-ADC3	
4001 1400 - 4001 17FF	USART6	
4001 1000 - 4001 13FF	USART1	
4001 0400 - 4001 07FF	TIM8	
4001 0000 - 4001 03FF	TIM1	
4000 7C00 - 4000 7FFF	UART8	APB1
4000 7800 - 4000 7BFF	UART7	
4000 7400 - 4000 77FF	DAC	
4000 7000 - 4000 73FF	PWR	
4000 6800 - 4000 6BFF	CAN2	
4000 6400 - 4000 67FF	CAN1	
4000 5C00 - 4000 5FFF	I2C3	
4000 5800 - 4000 5BFF	I2C2	
4000 5400 - 4000 57FF	I2C1	
4000 5000 - 4000 53FF	UART5	
4000 4C00 - 4000 4FFF	UART4	
4000 4800 - 4000 4BFF	USART3	
4000 4400 - 4000 47FF	USART2	
4000 4000 - 4000 43FF	I2S3ext	
4000 3C00 - 4000 3FFF	SPI3/I2S3	
4000 3800 - 4000 3BFF	SPI2/I2S2	
4000 3400 - 4000 37FF	I2S2ext	
4000 3000 - 4000 33FF	IWDG	
4000 2C00 - 4000 2FFF	WWDG	
4000 2800 - 4000 2BFF	RTC & BKP Reg	
4000 2000 - 4000 23FF	TIM14	
4000 1C00 - 4000 1FFF	TIM13	
4000 1800 - 4000 1BFF	TIM12	
4000 1400 - 4000 17FF	TIM7	
4000 1000 - 4000 13FF	TIM6	
4000 0C00 - 4000 0FFF	TIM5	
4000 0800 - 4000 0BFF	TIM4	
4000 0400 - 4000 07FF	TIM3	
4000 0000 - 4000 03FF	TIM2	

Informationen måste finnas för att vi exempelvis ska kunna implementera ”drivrutiner” för enheterna

Compiler library

gencode.c

```
int     a,b,c;
void    f( void )
{
        a = b/c;
}
```

gencodeV7.dass (arm**v7-m**)

```
  ldr   r2,b
  ldr   r3,c
  sdiv  r3,r2,r3
  ldr   r2,=a
  str   r3,[r2]
```

gencodev6.dass (arm**v6-m**)

```
  ldr   r0,b
  ldr   r1,c
  bl    __divsi3
  ldr   r3,=a
  str   r0,[r3]
```

lib/gcc/arm-none-eabi/.../...libgccc.a

```
10001010011100011001010
__divsi3:
00101100001010001001001001010
10001010011100011.....
```

Funktionerna i kompilatorbiblioteket används bara internt för kodgenerering, dvs. är ej publika och det behövs därför ingen header-fil med deklarationer.
Information om processorarkitektur måste vara konsistent vid kompilering och länkning.

Compiler library

I Grundläggande datorteknik har vi studerat algoritmer för multiplikation och division. Dessa kan användas för mjukvaruimpementering av operationerna.

EXEMPEL **4.39** ROBERTSONS METOD FÖR MULTIPLIKATION AV TAL MED TECKEN, X<0, Y<0.

Algoritm: Multiplikation P(produkt) = X(multiplikand) × Y (multiplikator)

Partialprodukt 0, PP(0) = 0

med början vid multiplikatorns LSB (y0)

för varje bit *i* hos multiplikatorn

Om y_i=1, addera multiplikand till nästa partialprodukt

annars addera 0 till nästa partialprodukt

skifta resultatet ett steg till höger

```
PP(3)   11101   110     ×2 (skifta aritmetiskt)
       +00110           y3=1⇒ADD -X
        00011   110
P=  30  =PP(4)=P   0001   1110
```

EXEMPEL **4.42** UTFÖR BINÄR DIVISION AV 13/5 MED ÅTERSTÄLLNINGSMETOD.

Svaret ska anges med 4 kvotbitar och 4 restbitar. Varje steg i algoritmen ska redovisas.

```
R(2) ×2=    0110100

+(-Y)       1011        steg 3: pröva
```

Algoritm: Division med återställning

$R = X - Q \times Y$

$Q=q_0q_1q_2...q_{n-1}$

n=antal kvotbitar att beräkna, partialrester R(i) bestäms enligt

R(0) = X

$R(1) = R(0) - q_0 \times Y$ $\quad q_0$=1 om R(1) ≥ 0, q_0 = 0 annars

för i = 2..n

$R(i)=2 \times R(i) - 1 - q_i * Y$ $\quad q_i$=1 om R(i) ≥ 0, q_i = 0 annars

I algoritmerna används enklare operationer: skift, addition och subtraktion.

Compiler library

Representation av flyttal

Ett flyttal uttrycks allmänt som:

$(-1)^S \ M \times 2^E$

där:

S (*sign*) är teckenbiten för flyttalet

S=0 anger ett positivt flyttal ty $(-1)^0 = 1$

S=1 anger ett negativt flyttal ty $(-1)^1 = -1$

M utgör talets mantissa

E utgör talets exponent.

Exponenten väljs från någon representation med inbyggt tecken. Det är värt att notera att för ett normaliserat flyttal på binär form gäller att:

$(M)_2 = 1.xxxxx$

dvs. mantissans första siffra är alltid är 1. Detta innebär att vi, då vi lagrar flyttal, kan utelämna denna siffra och på så vis åstadkomma ett kompaktare format.

Exempel: Omvandling av heltal till IEEE flyttal

Vi vill skriva talet $(2,52)_{10}\ 10^4$ som ett IEEE754-single format flyttal:

Vi har tidigare kommit fram till resultatet:

$(2,52)_{10}\ 10^4 = \quad = (1.100\ 0100\ 1110\ 000)_2 \times 2^{14}$

Mantissan ska ha totalt 24 bitar, vi ”fyller på” med nollor på slutet...

$M = (1.100\ 0100\ 1110\ 0000\ 0000\ 0000)_2$

Signifikanden F dvs. den del av mantissan som ska lagras får stryker den mest signifikanta ettan, vi har då:

$F = (100\ 0100\ 1110\ 0000\ 0000\ 0000)_2$

Exponenten i IEEE-formen uttrycks av karakteristikan E', excess(127) kod, dvs. $E' = E+127$, där E betecknar exponenten i talet vi utgår från (2^{14}) dvs. $E=14$ varför

$E' = 14 + 127 = 141 = (1000\ 1101)_2$

eftersom talet är positivt får vi $S = 0$. Vi sammanställer nu resultatet i 32-bitars form (*SFP*) och får slutligen:

$(2,52)_{10}\ 10^4 = (0100\ 0110\ 1100\ 0100\ 1110\ 0000\ 0000\ 0000)_{SFP}$

32bit single-precision:
- Sign: 1bit
- Mantissa: 24bits
- Exponent: 8bits

Representationen av heltal och flyttal är olika men de lagras med samma storlek.

```
float  f;     int i; /* båda typerna är 32 bitar */
```

Det innebär exempelvis att tilldelningen:

```
f = (float) i;
```

ska göras enligt exemplet ovan...

Compiler library

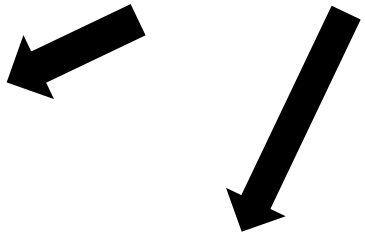
Flaggor till kompilatorn:

```
-mthumb;-march=armv6-m;-msoft-float
```

ger koden:

```
        ldr     r0,i
        bl      __aeabi_i2f
        ldr     r1,=f
        str     r0,[r1]
```

```
float  f;
int i;
        ...
        f = (float) i;
```



Flaggor till kompilatorn:

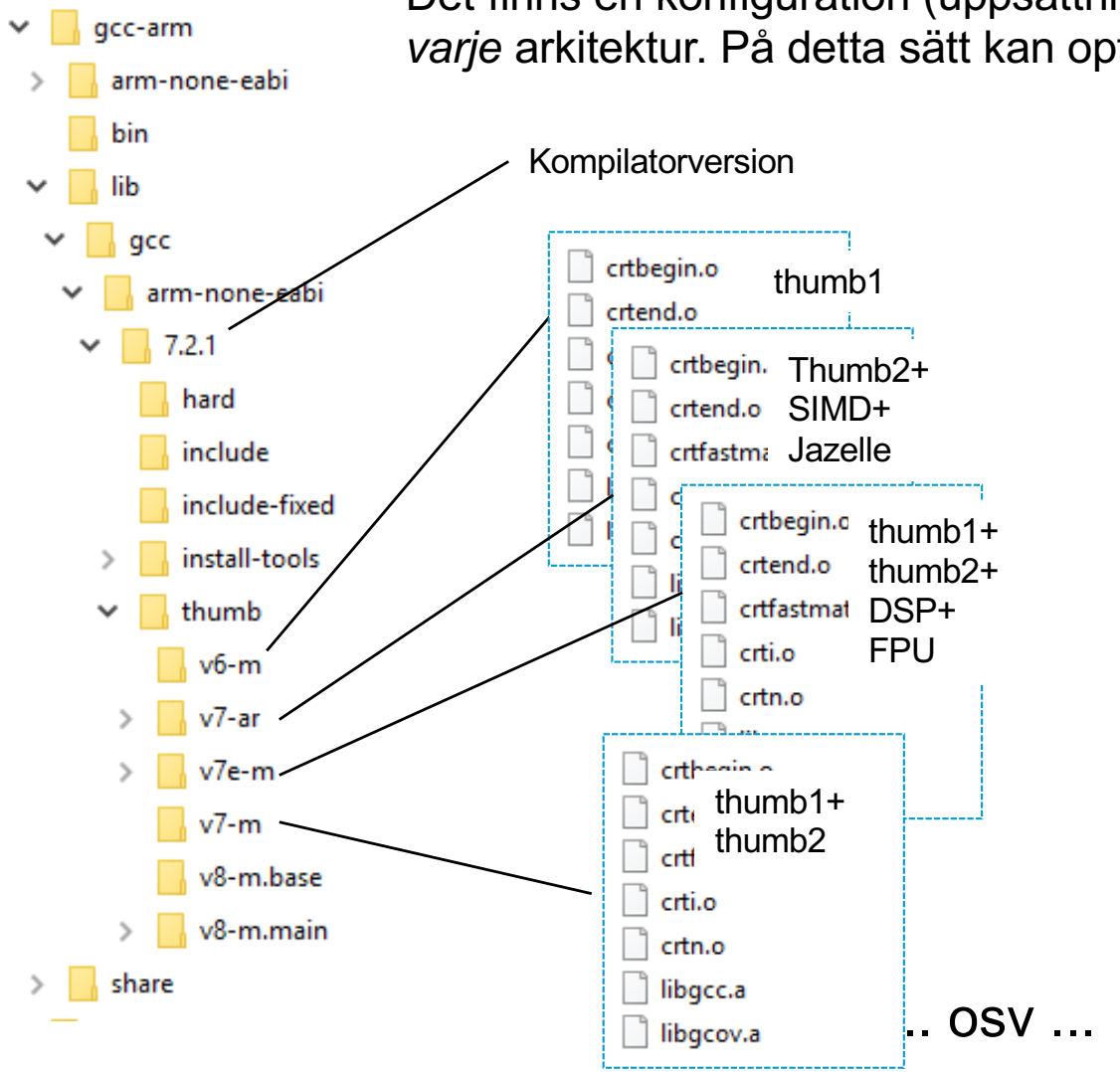
```
-mfloat-abi=hard;-mthumb;-mfpu=fpv4-sp-d16;-march=armv7-m;
```

ger koden:

```
ldr                 r0,i
vmov                s15,r0
vcvt.f32.s32        s15,s15
ldr                 r0,=f
vstr.32             s15,[r0]
```

Compiler library - different configurations

Det finns en konfiguration (uppsättning kompilator-bibliotek och startfiler) för *varje* arkitektur. På detta sätt kan optimal kod länkas till applikationen.

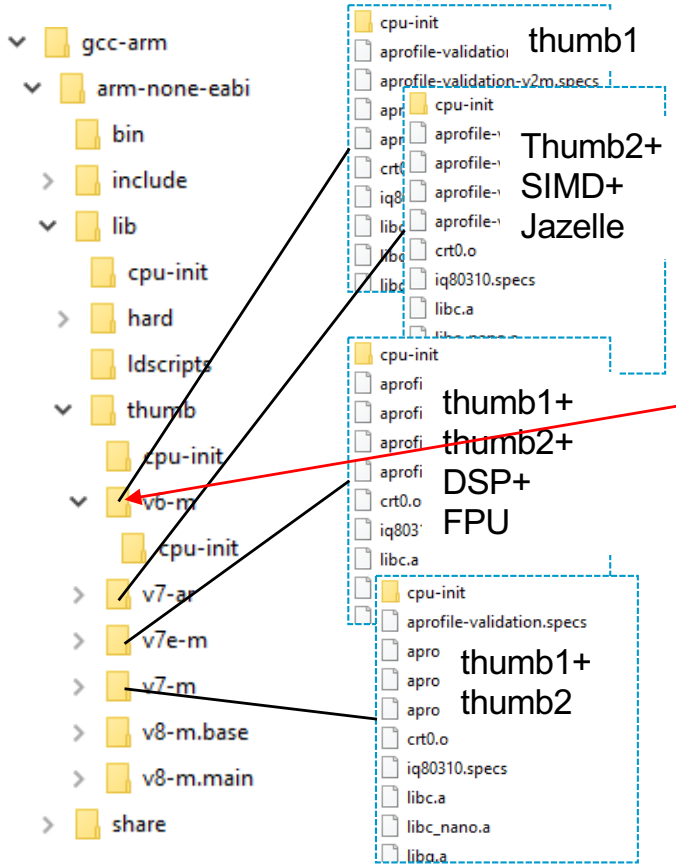


RTE - Run Time Environment
crtbegin, crtend etc..
Kallas också ”startfiles”

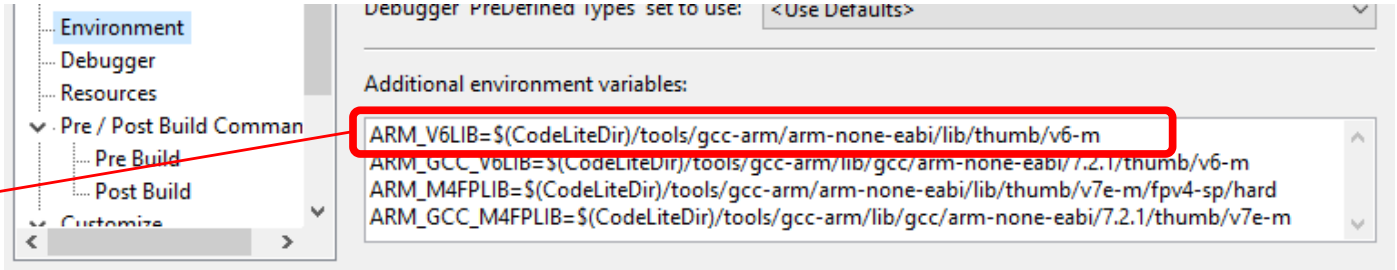
Libgcc, kompatibilitetskod

För att *undvika* länkning med standard filer ger man länkarflaggan **–nostartfiles**

C-library – different configurations



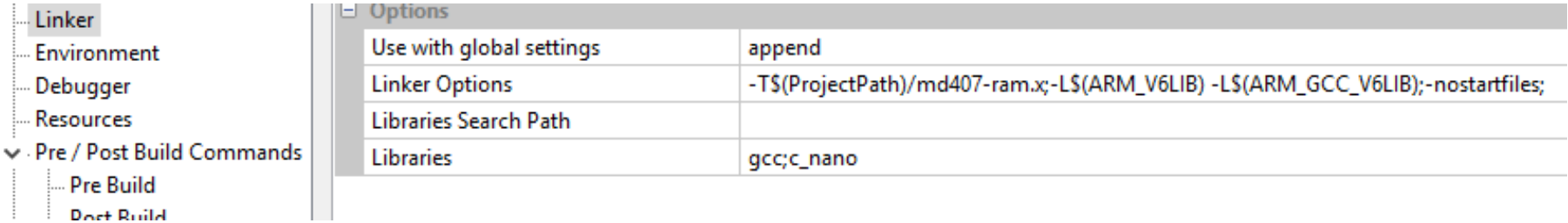
På samma sätt finns det en konfiguration (uppsättning C-bibliotek) för *varje* arkitektur. Vi använder miljövariabler för att precisera biblioteken:



I kommandoraden för länkaren kan vi sedan enkelt välja konfiguration.
I listan av bibliotek utelämnar man (konvention) delar av namnet, dvs:

`lib`**`gcc`**`.a` skrivs `gcc`
`lib`**`c_nano`**`.a` skrivs `c_nano`

Vi väljer också att använda vår egen ”run-time-miljö” **(-nostartfiles)**



Standard C library

"The C standard library" eller libc utgör standardbiblioteket för programspråket C och specificerades ursprungligen i ANSI C standarden från 1989, (C89).
Flera tillägg har gjorts efter detta.

<assert.h>	Definierar macro som kan användas för att upptäcka logiska fel och andra felaktigheter I "debug"-versioner av program.
<ctype.h>	Deklarerar funktioner för att klassificera och omvandla tecken.
<errno.h>	För att kontrollera felkoder från biblioteksfunktionerna.
<float.h>	Definierar olika konstanter som definierar implementationsspecifika detaljer i flyttalshanteringen.
<limits.h>	Definierar olika konstanter som definierar implementationsspecifika detaljer i heltalshanteringen.
<locale.h>	Deklarerar funktioner för lokala anpassningar.
<math.h>	Deklarerar vanliga matematiska funktioner..
<setjmp.h>	Deklarationer för speciella programflödesändringar.
<signal.h>	Deklarerar funktioner för hantering av "signaler".
<stdarg.h>	För variabelt antal argument till en funktion.
<stddef.h>	Definierar diverse olika typer och macron.
<stdio.h>	Deklarerar in- och utmatningsfunktioner.
<stdlib.h>	Deklarerar funktioner för numeriska konverteringar, slumptalsgenerering och minneshantering.
<string.h>	Deklarerar funktioner för stränghantering.
<time.h>	Deklarerar funktioner för datum och tid.

Dynamic memory management, malloc/free

C-biblioteket tillhandahåller rutiner som `malloc` och `free` för dynamisk minneshantering men har ingen information om hur måldatorns minne disponeras. Runtime biblioteket måste därför definiera:

```
void *_sbrk (int increment)
```

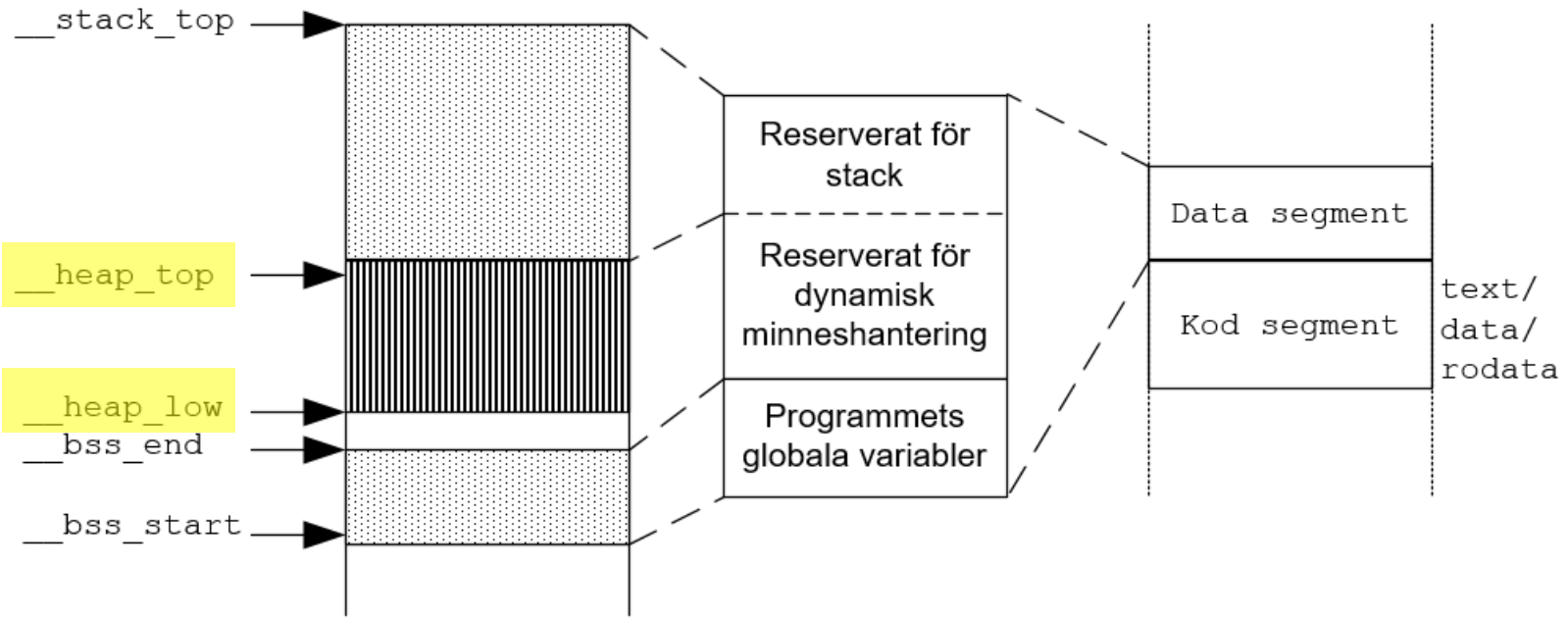
(*set program break*) som tillhandahåller adresser till minne som är tillgängligt för `malloc`.

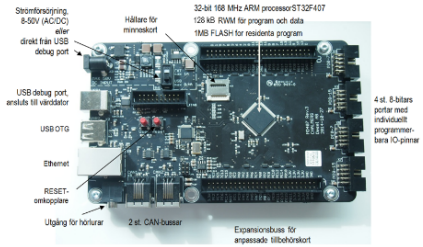
Länkaren skapar de symboler vi behöver för att administrera minnet

```
    ...
    *(.start_section)

    *(.text)

    *(.data)
    *(.rodata)
    . = ALIGN(4);
    __bss_start = .;
  *(.bss)
    __bss_end = .;
    . = ALIGN(4096);
    __heap_low = .;
    . = . + 0x400;
    __heap_top = .;
    . = . + 0x400;
    __stack_top = .;
```





Implementation of modified startup

Vår startup-sekvens för *md407* behöver modifieras som förberedelse för att använda C-biblioteket.

För applikationsprogram krävs att C-biblioteket initierats av någon funktion som utförts före **`main`**.
Standardprocedurer för *pre main* och *after main* finns `crt` (*c-run time)*.
Detta motsvaras av den ”startup” vi själva tidigare skapat i våra program.

```
...
   *(.start_section)
   *(.text)
   *(.data)
   *(.rodata)
   . = ALIGN(4);
   __bss_start__ = .;
   *(.bss)
   __bss_end__ = .;
   . = ALIGN(4096);
   __heap_low = .;
   . = . + 0x400;
   __heap_top = .;
   . = . + 0x400;
   __stack_top = .;
```

```
void startup ( void )
{
 __asm volatile(" LDR  R0,=__stack_top\n”);
 __asm volatile(" MOV  SP,R0\n");
 __asm volatile(" BL   crt_init\n");
 __asm volatile(" BL   main\n");
 __asm volatile(" BL   crt_deinit\n");
 __asm volatile("exit: B exit\n”);
}
```

Initiation of runtime functions

Standarden säger att icke initierade variabler (dvs. bss-arean) ska initieras till 0 av run-time systemet...

```
...
   *(.start_section)
   *(.text)
   *(.data)
   *(.rodata)
   . = ALIGN(4);
   __bss_start = .;
   *(.bss)
   __bss_end = .;
   . = ALIGN(4096);
   __heap_low = .;
   . = . + 0x400;
   __heap_top = .;
   . = . + 0x400;
   __stack_top = .;
```

```
void crt_init() {
  extern char __bss_start;
  extern char __bss_end;
  extern char __heap_low;
  extern char __heap_top;
  char *s;
  s = &__bss_start;
  while( s < &__bss_end)
    *s++ = 0;
  s = &__heap_low;
  while( s < &__heap_top )
    *s++ = 0;
}
```

Do you need to do this?

Implementation of _sbrk

```
#include      <errno.h>
static char *heap_end = 0;
char * _sbrk( int incr) {
  extern char __heap_low;
  extern char __heap_top;
  char *prev_heap_end;
  if (heap_end == 0) {
    heap_end = &__heap_low;
  }
  prev_heap_end = heap_end;

  if (heap_end + incr > &__heap_top) {
    /* Heap and stack collision */
    errno = ENOMEM;
    return (char *) 0;
  }
  heap_end += incr;
  return (char *) prev_heap_end;
}
```

```
...
   *(.start_section)

   *(.text)

   *(.data)
   *(.rodata)
   . = ALIGN(4);
   __bss_start = .;
  *(.bss)
   __bss_end = .;
   . = ALIGN(4096);
   __heap_low = .;
   . = . + 0x400;
   __heap_top = .;
   . = . + 0x400;
   __stack_top = .;
```

Use of the standard C-library

Typdeklarationer för funktionerna i C-biblioteket har organiserats i olika header-filer.

main.c

```
#include <stdio.h>  /* printf */
#include <stdlib.h> /* malloc */

void foo(){
  char *cp;
  cp = (char *) malloc( 100 );
  if( cp == (char *) 0
  {
    printf( "Can't mall  );
    exit(-1);
  }
  free( cp );
}
```

malloc() is at the user-level and sbrk() is at the kernel level

Q1: malloc(10) calls sbrk(19?
Q2: free(cp) calls sbrk(x)?

arm-none-eabi/include/stdio.h

```
#ifndef _STDIO_H_
#define _STDIO_H_
...
int printf(const char *,...);
...
```

arm-none-eabi/lib/..../libc.a

```
malloc:
10001010011100011001O1 ....
printf:
00101100001010001001001O010
1000101001110001l.....
free: 10100111000110 ....
```

"C-nano" – ligh

Applikation...

```
#include <stdio.h>

void main(void)
{
  printf( "Hello World!");
}
```

CHALMERS
UNIVERSITY OF TECHNOLOGY

File operations

```
#include <stdio.h>

#define MAX 100

int main(void) {
  FILE *fp;
  char *filename = "x1.c";
  char str[MAX];
  int linenum = 0;

  fp = fopen(filename, "r");
  if( fp == NULL ) {
    printf( "ERROR: Could not find file %s\n", filename );
  }
  while(!feof(fp)) {
    fgets( str, MAX, fp );
    printf("%d: %s", linenum++, str);
  }
  printf("\n");
  fclose(fp);
}
```

5/2/19

Då vi länkar vår applikation upptäcker vi att en rad symboler saknas.
Detta beror på att IO-funktioner i standard C-biblioteket också länkas mot *maskinberoende* funktioner (C-RunTime), som:
`_open, _close, _lseek, _read, _write, _fstat, _isatty`

För att kunna använda IO-funktionerna måste vi då först implementera dessa maskinberoende funktioner.

stdio – keypad and ASCII-display

Data structure: ”Device driver”.

```
#include <sys/stat.h>

typedef struct
{
  char   name[16];          /* Device name */
  int (*init) (int);
  void (*deinit) (int);
  int (*fstat)(struct stat *st);
  int (*isatty)(void);
  int (*open)(const char name,int flags,int mode);
  int (*close)(void);
  int (*lseek)(int ptr, int dir);
  int (*write)(char *ptr, int len);
  int (*read)(char *ptr, int len);
} DEV_DRIVER_DESC, *PDEV_DRIVER_DESC;
```

```
static DEV_DRIVER_DESC KeyPad =
{
  {"Keypad"},
  keypad_init,
  keypad_deinit,
  0,
  0,
  0,
  0,
  0,
  0,
  keypad_read
};
```

stdio – keypad and ASCII-display

”Device driver”.

```
#define MAX_FILENO          5
#define KEYPAD_FILENO       3
#define ASCII_DISPLAY_FILENO        4

PDEV_DRIVER_DESC open_file_table[MAX_FILENO] =
{
        &StdIn,
        &StdOut,
        &StdErr,
        &KeyPad,
        &AsciiDisplay
};

void crt_init() {
  …
  for( int i = 0; i < MAX_FILENO; i++ )
  {
    fd = open_file_table[i];
    if( fd->init != 0)
      (void) fd->init( 0 );
   }
}
```

```
static DEV_DRIVER_DESC KeyPad =
{
  {"Keypad"},
  keypad_init,
  keypad_deinit,
  0,
  0,
  0,
  0,
  0,
  0,
  keypad_read
};
```

```
#include <sys/stat.h>

typedef struct
{
  char      name[16];   /* Device name */
  int (*init) (int);
  void (*deinit) (int);
  int (*fstat)(struct stat *st);
  int (*isatty)(void);
  int (*open)(const char name,int flags,int mode);
  int (*close)(void);
  int (*lseek)(int ptr, int dir);
  int (*write)(char *ptr, int len);
  int (*read)(char *ptr, int len);
} DEV_DRIVER_DESC, *PDEV_DRIVER_DESC;
```

"C-run time" - lightweight implementation for MD407

`_fstat`, returnera information om en öppen fil.

```
#include <sys/stat.h>
int _fstat(int file, struct stat *st) {
 st->st_mode = S_IFCHR;
 return 0;
}
```

`struct stat` och `S_IFCHR` deklareras i `sys/stat.h`

`S_IFCHR`, anger att detta är en tecken orienterad fil, andra exempel är:
`S_IFDIR`, filen är ett bibliotek.
`S_IFBLK`, filen är block-orienterad, (typiskt på disk) osv...
Vår implementering kommer endast att stödja de speciella filerna `stdin`, `stdout` och `stderr`, som teckenorienterade filer, dvs. in- och utmatning via en terminal.

```
int _isatty(int file) { return 1; }
```

`stdin`, `stdout` och `stderr`, kan normal sett dirigeras om till blockorienterade enheter, dock inte i vår implementering...

"C-run time" – lightweigth implementation for MD407

`_open`, öppna en fil för läsning och/eller skrivning.

I vår implementering gör vi det enkelt och låter systemet öppna filer för enheterna, det räcker då att vår implementering returnerar en felkod.

```
int _open(const char *name, int flags, int mode) { return -1; }
```

`_close`, stäng en fil som är öppen för läsning och/eller skrivning.

Ej tillämpbart, våra öppna enheter stängs av operativsystemet.

```
int _close(int file) { return -1; }
```

`_lseek`, sök till position i en fil som är öppen för läsning och/eller skrivning.

Måste vara en blockorienterad fil, ej tillämpbart för teckenorienterade enheter.

```
int _lseek(int file, int ptr, int dir) { return 0; }
```

"C-run time" – lightweigth implementation for MD407

`_write`, skriv till en öppen fil.

Vår implementering är enklast tänkbara . Man hade kunnat lägga in felkontroll här eftersom `file` ska vara någon av `stdout` eller `stderr`....

```
int _write(int file, char *ptr, int len) {
  PDEV_DRIVER_DESC drvr;
  drvr = open_file_table[file];
  if(drvr == 0)
    return 0;
  return drvr->write(ptr,len);
}
```

`_read`, läs från en öppen fil.

```
int _read(int file, char *ptr, int len) {
  PDEV_DRIVER_DESC drvr;
  drvr = open_file_table[file];
  if(drvr == 0)
    return 0;
  return drvr->read(ptr,len);
}
```

```
static DEV_DRIVER_DESC KeyPad =
{
  {"Keypad"},
  keypad_init,
  keypad_deinit,
  0,
  0,
  0,
  0,
  0,
  0,
  keypad_read
};
```

Create a run-time library for MD407

EXEMPEL

För att slippa kompilera vårt lilla runtime-bibliotek *tillsammans* med applikationen skapar vi i stället ett programbibliotek `libMD407.a`

```
/*
 * libMD407.c
 * MD407 library
 */

/* declarations goes to  'libMD407.h' */
#include    "libMD407.h"

/* Define variables here */
static char *heap_end;

/* Define functions here */
void crt_init() {
...
```

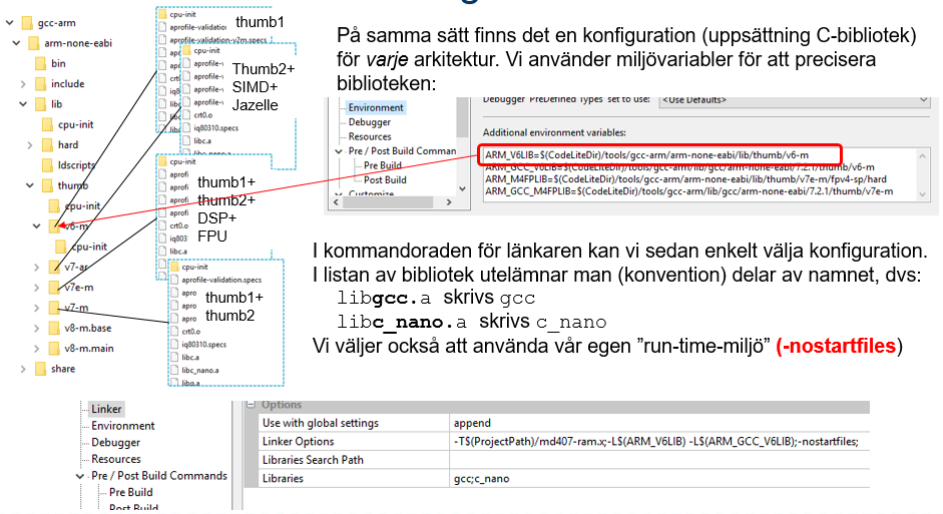
```
/*
* libMD407.h
* Declaration of library functions, constants etc...
*/
#include    <stdio.h>
#include    <errno.h>
#include    <sys/stat.h>
/* Type definitions */
typedef struct
{
 ...
  volatile unsigned short sr;
} DEV_DRIVER_DESC, *PDEV_DRIVER_DESC;
/* Constants */
#define MAX_FILENO      5
/* Constant defined by linker */
extern char __heap_low;
extern char __heap_top;
extern char __bss_start;
extern char __bss_end;
/* Library defined functions */

void crt_init(void);
void crt_deinit( void );

char *_sbrk(int);
int _close(int file);
int _fstat(int file, struct stat *st);
etc ...
```

`libMD407.a`, installation and use

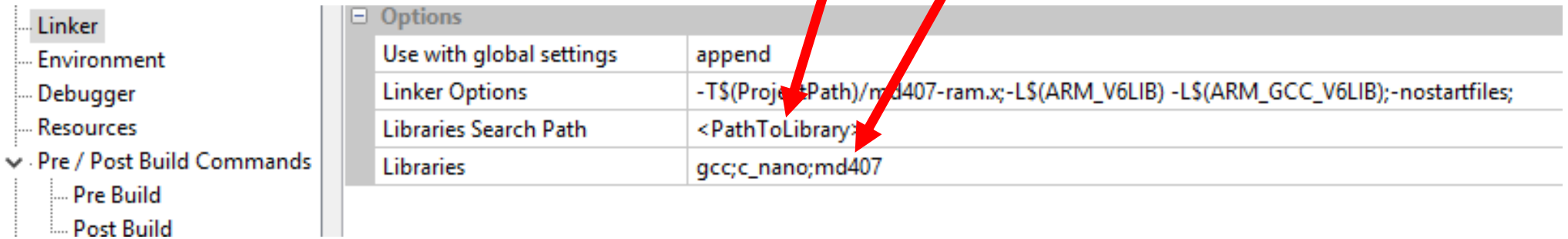
C-bibliotek – olika konfigurationer



Sökväg till det nya biblioteket.

Namn på nytt bibliotek.

Man kan välja att installera biblioteket i en befintlig sökväg, eller att skapa en ny. I vilket fall, måste man lägga till programbiblioteket i listan av bibliotek:



Use it easily now!

```
void main(void)
{
        char    buffer[256];
        char *dbuf1,*dbuf2,*dbuf3;
        dbuf1 = malloc( 100);
        dbuf2 = malloc( 150);
        dbuf3 = malloc( 15);
        free( dbuf2);
        dbuf2 = malloc( 50);

        printf( "libc_nanotest");
        fflush( stdout );
        while( 1 )
        {
                getline( buffer, 256 );
                fflush( stdin );

                printf("\n%s", buffer);
                fflush( stdout );
        }
}
```