

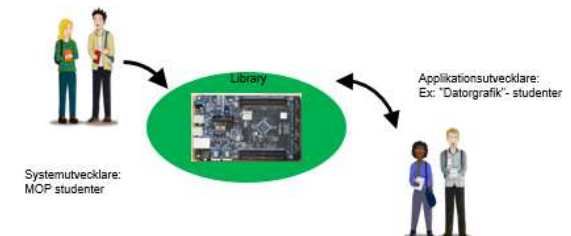
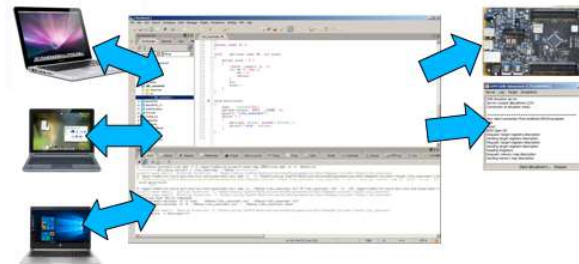
Programbibliotek

Ur innehållet:

Exekveringsmiljö

Olika typer av programbibliotek

- Kompilatorbibliotek
- C-bibliotek



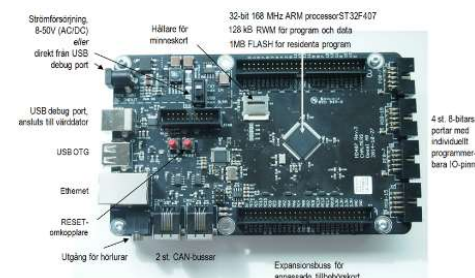
Läsanvisningar:

Arbetsbok kapitel 8

Målsättningar:

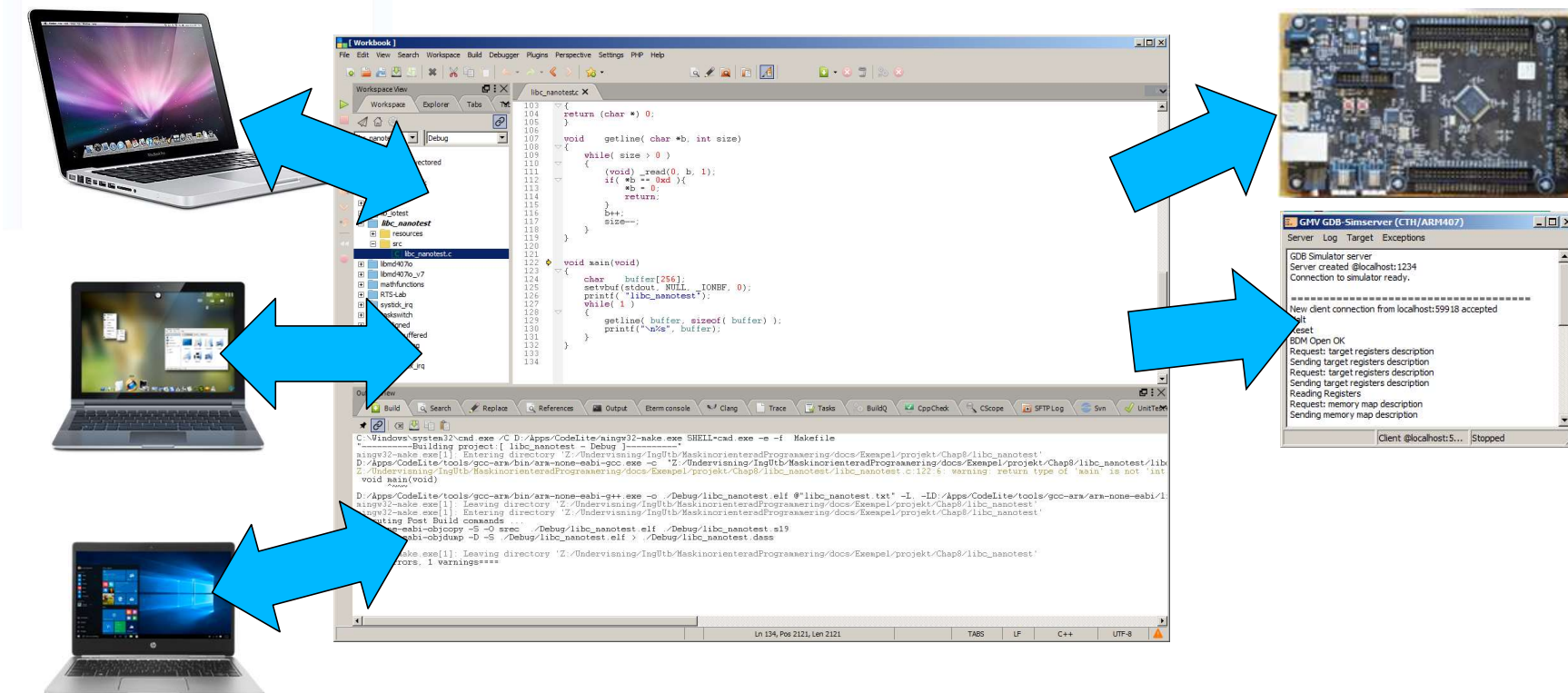
Att kunna använda färdiga programbibliotek

Att kunna skapa och underhålla ett nytt programbibliotek



Värddator och måldator:

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



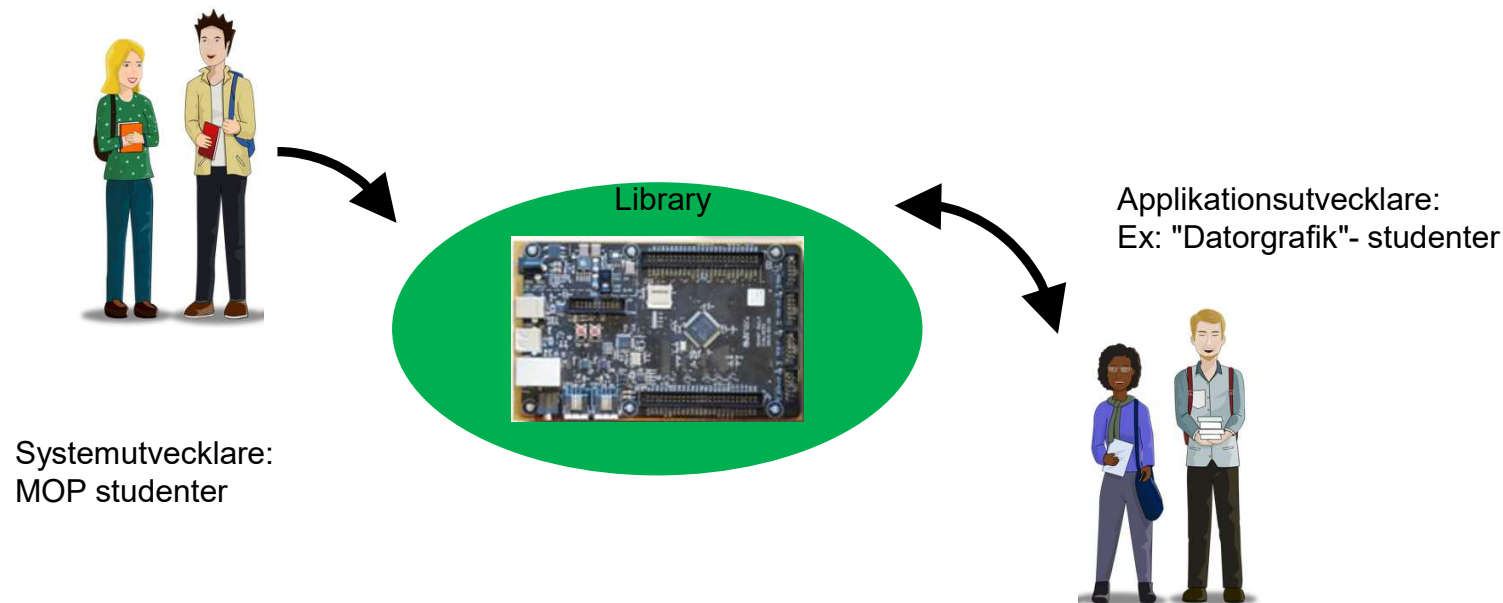
System-/Applikations- programmering för MD407

Med *programbibliotek*, avser vi att skapa grundläggande funktionalitet för användningen av vårt datorsystem. *Exempel: implementering av grundläggande operationer i programvara. In-/ut-matning, filhantering, fönsterhantering etc.*

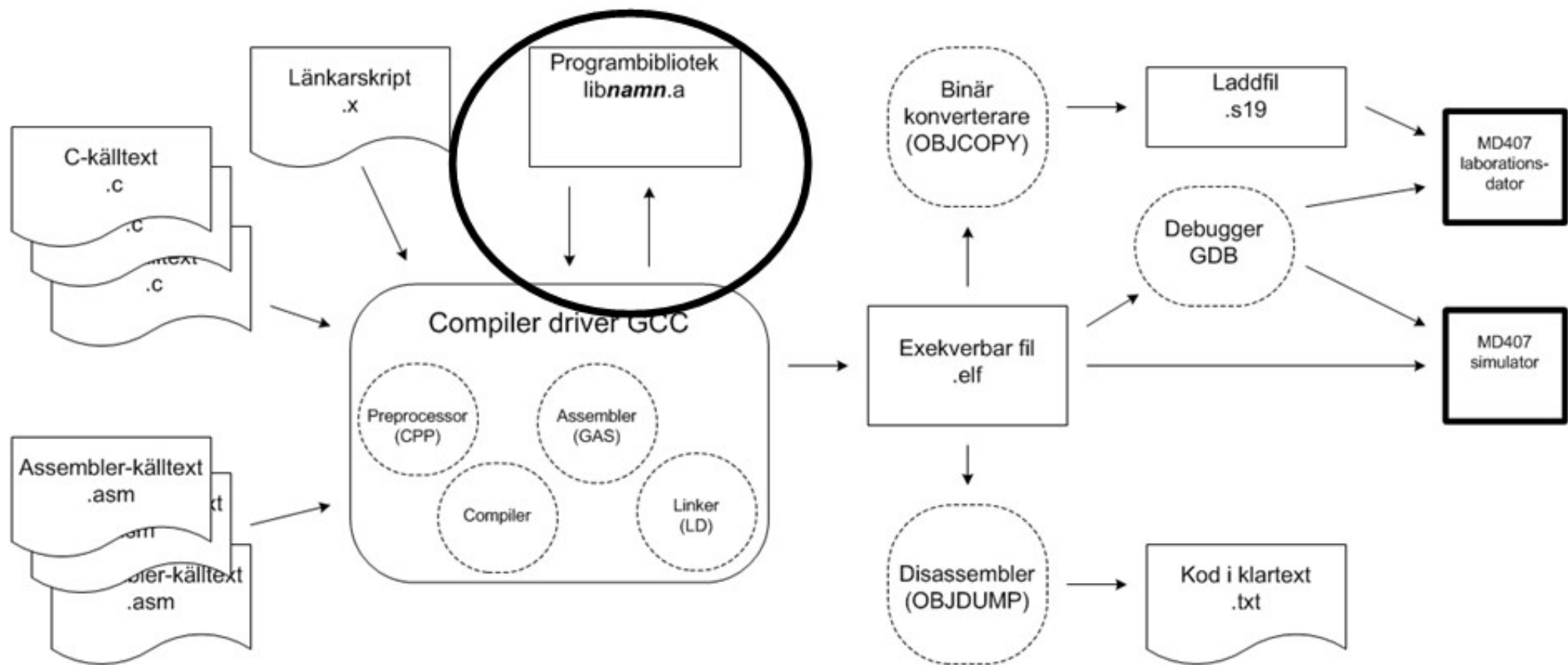
Med *systemprogramvara* menar vi oftast "basal" programvara för vår dator, ett programlager med generellt användbar natur. *Exempel: operativsystem och drivrutiner.*

Med *applikationsprogramvara*, menar vi program avsedda för mer specifik användning.

Exempel: Spel, navigation, övervakning/styrning etc.



Programbibliotek



Programbibliotek – olika typer

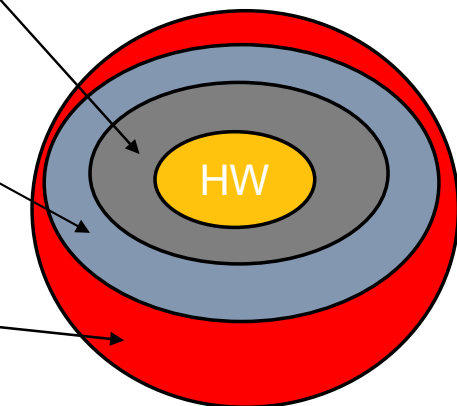
Programbibliotek kan ofta ses som "kompatibilitetslager" där en anpassning sker till någon speciell maskinvara och något specifikt operativsystem.

- 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.**

Exempel:
Flyttalsoperationer

Exempel:
malloc/free, printf...

Exempel: GUI: Fönster
och menyer

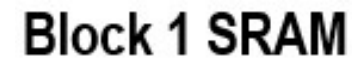


[illegible]

- Arkitektur – ARM-Cortex, tillhandahåller processorer som hanterar olika typer av instruktionsuppsättningar:

- | Instruktion | Storek | Cortex M0 | Cortex M0+ | Cortex M1 | Cortex M3 | Cortex M4 | Cortex M7 | Ark. |
|---|--------|-----------|------------|-----------|-----------|-----------|-----------|--------------|
| ADC, ADD, ADR, AND, ASR, B, BIC, BKPT, BLX, BV, CMN, CMP, CPS, EOR, LDM, (LDMA, LDMPD),
LDR, LDRB, LDRH, LDRSB, LDRSH, LSL, LSR, MOV, MVN, NOP, ORR, POP, PUSH, REV,
REV16, REVSH, ROR, RSB, SBC, SEV, STM, (STMA, STMEA), STR, STRB, STRH, SUB, SVC, SXTB,
SXTH, STXTH, UXTB, WFE, WFI, YIELD | 16-bit | x | x | x | x | x | x | v6 |
| B.DMB, DBZ, 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, BFJ, BIC, CDOP, CLREX, CLZ, CMN, CMP, DSB, EOR, LDC, LDMA,
LDMPB, LDR, LDRB, LDRGT, LDRH, LDREX, LDREXB, LDREXH, LDRH, LDRHT, LDRSB, LDRSBT,
LDRT, LDRH, LDRH, LSL, LSR, MLS, MORR, MA, MOV, MOV7, MR, MRRC, MOV, MVN, NOP,
ORR, ORR, PLD, PLDW, PUI, POP, PUSH, RBIT, REV, REV16, REVSH, ROR, RRX, RSB, SCB, SBFX,
SDIV, SEV, SMLAL, SMLALT, SSAT, CT, STMDB, STR, STRB, STRD, STRDT, STREX, STREXB,
STREXH, STRH, STRHT, STRT, SUB, SXTB, SXTB, TBB, TBH, TEQ, TST, UBFX, UDIV, UMIAL,
UMIUB, USAT, UXTB, UXTB, WFE, WFI, YIELD | 32-bit | | | | x | | x | v7 |
| PKH, QADD, QADDD, QADDS, QASX, QDADD, QDSUB, QSAX, QSUB, QSUBBS, QSUBB, SADDB, SADD16,
SADDB, SASX, SEL, SHADD16, SHADDS, SHASX, SHSUB16, SHSUBB, SMLABB, SMLABT,
SMILATE, SMILT, SMUL, SMLALB, SMLALBT, SMLATB, SMLATBT, SMLD, SMLD, SMLALB, SMLALB,
SMLSD, SMLS, SMLLA, SMLLS, SMMUL, SMUAD, SMLLB, SMLLT, SMLTT, SMLTBT,
SMULT, SMULVB, SMUSD, SSAT16, SSAB, SSUB16, SSUBB, SXTAB, SXTAB, SXTB16, SXTBH, SXTB16,
UADDD16, UADDB, UASX, UHADDD16, UHADDB, UHASX, UHSUB16, UHSUBB, UMAL,
UQADDD16, UQADDB, UQASX, UQSAX, UQSUB16, UQSUBB, UQSUB, USAD, USADB, USAT16, USAX,
USAXB, USUB, UXTAB, UXTAB, UXTB, UXTB | 32-bit | | | | x | x | | v7e
(DSP) |
| VABS, VADD, VCMP, VCMPLE, VCVT, VCVTB, VDIV, VLDM, VLDR, VMLA, VMLS, VMOV, VMRS,
VMUL, VMSR, VNEG, VNMLA, VNMLB, VNMLH, VNMLL, VQDIB, VQDIBH, VQDOT, VQSTM, VQSTB, VQSUB, | 32-bit | | | | | | SP
FPU | 32-b |
| FP - dubbel precision | 32-bit | | | | | | DP
FPU | 64-b |

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



```

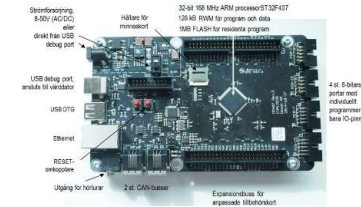
2001 FFFF
2001 C400
2001 C3FF
2001 C000
2001 BFFF

```

Monitor/
debugger
stack och data
Relokerade
avbrottsvektorer

Reserverat för applikationen

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



Måldator – periferienheter

- Tillgängliga periferienheter –
Systemenheter
IO-enheter

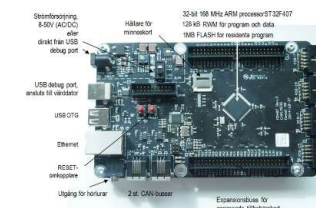
Adressrum	
Tillverkar-specifikt	FFFF FFFF
	E010 0000
	E00F FFFF
Periferibuss	
	E000 0000
	DFFF FFFF
Externa enheter	
	A000 0000
	9FFF FFFF
Extern RAM-minne	
	6000 0000
	5FFF FFFF
Periferekretsar	
	4000 0000
	3FFF FFFF
Block 1 SRAM	
	2000 0000
	1FFF FFFF
Block 0 kod	
	0000 0000

Periferibuss

E000 EF44	"Floating point unit"
E000 EF30	"NVIC"
E000 EF03	"Memory protection unit"
E000 EDB8	"Floating point unit, access control"
E000 ED8B	"System Control Block"
E000 ED88	"NVIC"
E000 ED3F	"System timer"
E000 ED00	
E000 E42F	
E000 E100	
E000 E01F	
E000 E010	

Periferekretsar

A000 0000 - A000 0FFF	FSMC control reg.	AHB3
5006 0800 - 5006 08FF	RNG	
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 8000 - 4002 80FF	DMA2D	
4002 9000 - 4002 93FF		
4002 8C00 - 4002 8FFF		
4002 8800 - 4002 8BFF		
4002 8400 - 4002 87FF		
4002 8000 - 4002 83FF		
4002 6400 - 4002 67FF		
4002 6000 - 4002 63FF		
4002 4000 - 4002 4FFF		
4002 3C00 - 4002 3FFF		
4002 3800 - 4002 3BFF		
4002 3000 - 4002 33FF		
4002 2800 - 4002 2BFF		
4002 2400 - 4002 27FF		
4002 2000 - 4002 23FF		
4002 1C00 - 4002 1FFF		
4002 1800 - 4002 1BFF		
4002 1400 - 4002 17FF		
4002 1000 - 4002 13FF		
4002 0C00 - 4002 0FFF		
4002 0800 - 4002 0BFF		
4002 0400 - 4002 07FF		
4002 0000 - 4002 03FF		
4001 6800 - 4001 6BFF		
4001 5800 - 4001 5BFF		
4001 5400 - 4001 57FF		
4001 5000 - 4001 53FF		
4001 4800 - 4001 4BFF		
4001 4400 - 4001 47FF		
4001 4000 - 4001 43FF		
4001 3C00 - 4001 3FFF		
4001 3800 - 4001 3BFF		
4001 3400 - 4001 37FF		
4001 3000 - 4001 33FF		
4001 2C00 - 4001 2FFF		
4001 2000 - 4001 23FF		
4001 1400 - 4001 17FF		
4001 1000 - 4001 13FF		
4001 0400 - 4001 07FF		
4001 0000 - 4001 03FF		
4000 7C00 - 4000 7FFF		
4000 7800 - 4000 7BFF		
4000 7400 - 4000 77FF		
4000 7000 - 4000 73FF		
4000 6800 - 4000 6BFF		
4000 6400 - 4000 67FF		
4000 5C00 - 4000 5FFF		
4000 5800 - 4000 5BFF		
4000 5400 - 4000 57FF		
4000 5000 - 4000 53FF		
4000 4C00 - 4000 4FFF		
4000 4800 - 4000 4BFF		
4000 4400 - 4000 47FF		
4000 4000 - 4000 43FF		
4000 3800 - 4000 3BFF		
4000 3400 - 4000 37FF		
4000 3000 - 4000 33FF		
4000 2C00 - 4000 2FFF		
4000 2800 - 4000 2BFF		
4000 2000 - 4000 23FF		
4000 1C00 - 4000 1FFF		
4000 1800 - 4000 1BFF		
4000 1400 - 4000 17FF		
4000 1000 - 4000 13FF		
4000 0C00 - 4000 0FFF		
4000 0800 - 4000 0BFF		
4000 0400 - 4000 07FF		
4000 0000 - 4000 03FF		



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

Kompilatorbibliotek

gencode.c

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

gencodeV7.dass (armv7-m)

```
LDR    R2,b
LDR    R3,c
SDIV   R3,R2,R3
LDR    R2,=a
STR    R3,[R2]
```

gencodev6.dass (armv6-m)

```
LDR    R0,b
LDR    R1,c
BL     __divsi3
LDR    R3,=a
STR    R0,[R3]
```

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

```
1000101001110001100101
__divsi3:
00101100001010001001001001000
1010011100011.....
```

Funktionerna i kompilatorbiblioteket används av kompilatorn 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.

Kompilatorbibliotek

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

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

Algoritm: Multiplikation $P(\text{produkt}) = X(\text{multiplikand}) \times Y(\text{multiplikator})$
 Partialprodukt 0, $PP(0) = 0$
 med början vid multiplikatorns LSB (y_0)
 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)	11101110	$\times 2$ (skifta aritmetiskt)
	+00110	$y_3 = 1 \Rightarrow \text{ADD } -X$
	00011110	
P = 30 = PP(4) = P	00011110	

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) $\times 2 =$	0110100
+ (-Y)	1011 steg 3: pröva

ikna
 } bitar ska skiftas):

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

Algoritm: Division med återställning

$R = X - Q \times Y$
 $Q = q_0 q_1 q_2 \dots q_{n-1}$
 $n = \text{antal kvotbitar att beräkna, partialrester } R(i) \text{ bestäms enligt}$
 $R(0) = X$
 $R(1) = R(0) - q_0 \times Y$ $q_0 = 1$ om $R(1) \geq 0$, $q_0 = 0$ annars
 för $i = 2 \dots n$
 $R(i) = 2 \times R(i-1) - q_{i-1} \times Y$ $q_i = 1$ om $R(i) \geq 0$, $q_i = 0$ annars

Kompilatorbibliotek

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 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:

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

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 stryka 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:

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

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

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

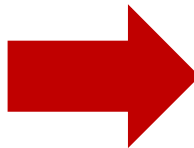
Det innebär exempelvis att tilldelningen:

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

ska göras enligt exemplet ovan...

Kompilatorbibliotek

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



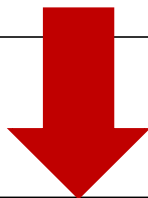
(armv6-m) exempelvis Cortex M3

Flaggor till kompilatorn:

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

ger koden:

```
LDR    R0, i  
BL    __aeabi_i2f  
LDR    R1, =f  
STR    R0, [r1]
```



(armv7-m + floating point unit) exempelvis Cortex M4

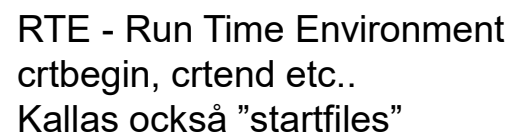
Flaggor till kompilatorn:

```
-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]
```

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



Libgcc, kompatibilitetskod

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

"C-nano" - lättviktsimplementering av Standard C

Applikation...

```
#include <stdio.h>

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

Standard C-bibliotek

libc_nano.a

```
100010100111000110101
printf:
001011000010100010010
010010100010100111000
11.....
```

C-RunTime, (CRT)
funktioner

```
open()
close()
write()
read(),
etc måste utformas baserat
på hårdvaran
```



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, i form av så kallade *drivrutiner*, för MD407.

Exempel: "Device driver" keypad och ASCII-display

Datastruktur: "Device driver".

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

typedef struct
{
    char name[16];
    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
};
```

```
static DEV_DRIVER_DESC AsciiDisplay =
{
    {"AsciiDisplay"},
    asciidisplay_init,
    asciidisplay_deinit,
    0,
    0,
    0,
    0,
    0,
    0,
    asciidisplay_write,
    0
};
```

Exempel: Tabell med "Device drivers"

```
#define MAX_FILENO 5

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

void crt_init() {
    ...
    for( int i = 0; i < MAX_DEVICE; i++ )
    {
        fd = device_table[i];
        if( fd->init != 0 )
            (void) fd->init( 0 );
    }
}
```

```
static DEV_DRIVER_DESC StdIn =
{
    {"stdin"},
    usart_init,
    usart_deinit,
    0,
    usart_isatty,
    0,
    0,
    0,
    0,
    usart_read
};
```

```
static DEV_DRIVER_DESC StdOut =
{
    {"stdin"},
    0,
    usart_deinit,
    0,
    usart_isatty,
    0,
    0,
    0,
    0,
    usart_write,
    0
};
```

"C-run time" - lättviktsimplementering för 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 teckenorienterade enheter.

_isatty ska returnera 1 om filen är av terminal-typ. stdin, stdout och stderr, kan normal sett dirigeras om till blockorienterade enheter, dock inte i vår implementering. isatty implementeras därför, men enbart för stdin, stdout och stderr.

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

"C-run time" - lättviktsimplementering för MD407

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

Ej tillämbart, vi har inget filsystem, våra enheter öppnas (`init`) av runtime-systemet. Det räcker då att vår implementering returnerar en felkod om `open` anropas.

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

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

Ej tillämbart, våra enheter stängs av runtime-systemet (`deinit`).

```
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 tillämbart för teckenorienterade enheter.

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

"C-run time" - lättviktsimplementering för MD407

_write, skriv till en öppen fil eller enhet.

Funktionen ska dirigera operationen till enhetens write-funktion om en sådan finns.

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

_read, läs från en öppen fil eller enhet.

Funktionen ska dirigera operationen till enhetens read-funktion om en sådan finns.

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

Skapa run-time bibliotek för MD407

Under laboration 6 ska du skapa ett programbibliotek `libmd407.a` som kompletterar med de hårdvaruberoende run-time funktionerna

```
/*
 * libmd407.c
 */
/*
 * driver_asciisp
 * MD407 library
 */
/*
 * driver_keypad
 * MD407 library
 */
/*
 * declarations
 */
#include "libmd407.h"
static int keypad_init(void);
static void keypad_deinit(void);
static int keypad_write(char *ptr, int len);
static int keypad_read(char *ptr, int len);

DEV_DRIVER_DESC StdIn =
{
    {"stdin"},
    keypad_init,
    keypad_deinit,
    keypad_write,
    keypad_read,
};

DEV_DRIVER_DESC StdOut =
{
    ...
};

DEV_DRIVER_DESC StdErr =
{
    ...
};

/* Define functions here */
void usart_init( int initval ) { ... }
void usart_deinit( int deinitval ) { ... }
void usart_isatty( void ) { ... }
int usart_write(char *ptr, int len) { ... }
int usart_read(char *ptr, int len) { ... }
```

```
/*
 * libmd407.h
 * Declaration of library functions, constants etc...
 */
#ifndef _LIBMD407_H
#define _LIBMD407_H
#include <stdio.h>
#include <errno.h>
#include <sys/stat.h>
#include <unistd.h>

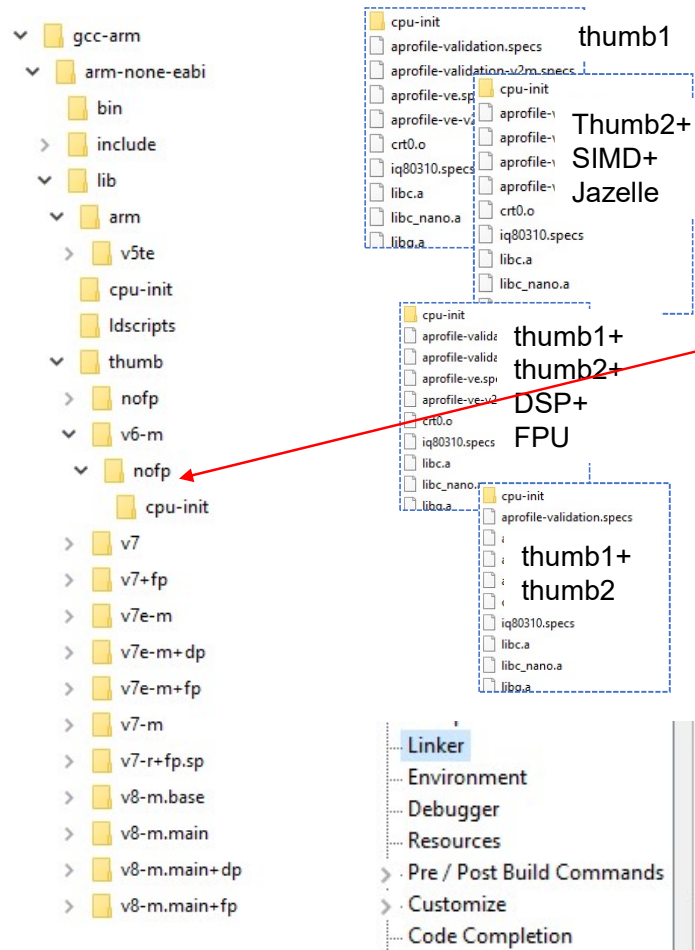
/* Type definitions */
typedef struct
{
    ...
} DEV_DRIVER_DESC, *PDEV_DRIVER_DESC;

/* 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__;

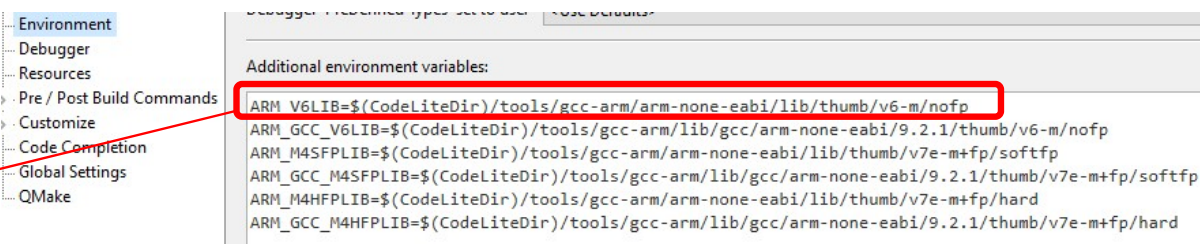
/* Library defined functions */
void crt_init( void );
void crt_deinit( void );

#endif /* LIBMD407_H */
```

C-bibliotek – olika konfigurationer



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



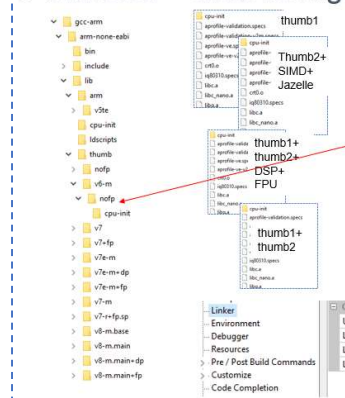
I kommandoraden för länkaren kan vi sedan enkelt välja konfiguration. I listan av bibliotek utelämnar man (konvention) delar av namnet, dvs:

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

Options	
Use with global settings	append
Linker Options	-T\$(ProjectPath)/md407-ram-crt.x;-L\$(ARM_V6LIB) -L\$(ARM_GCC_V6LIB);-nostdlib;-nostartfiles;--specs=nano.specs
Libraries Search Path	
Libraries	c_nano;md407;gcc;c_nano

libmd407.a, installation och användning

C-bibliotek – olika konfigurationer



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

```

Additional environment variables:
ARM_V6LIB=$(CodeLiteDir)/tools/gcc-arm-none-eabi/11.3/thumbv7e-m/nofp
ARM_THUMBLIB=$(CodeLiteDir)/tools/gcc-arm-none-eabi/11.3/thumbv7e-m/nofp
ARM_GCC_NANO=$(CodeLiteDir)/tools/gcc-arm-none-eabi/11.3/thumbv7e-m/nofp
ARM_THUMBFP=$(CodeLiteDir)/tools/gcc-arm-none-eabi/11.3/thumbv7e-m/fp/hard
ARM_GCC_NANOFPU=$(CodeLiteDir)/tools/gcc-arm-none-eabi/11.3/thumbv7e-m/fp/hard
  
```

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

```

libgcc.a skrivs gcc
libc_nano.a skrivs c_nano
  
```

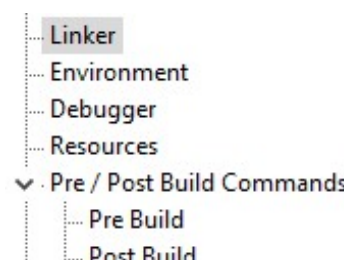
Options

Use with global settings	append
Linker Options	-T\$(ProjectPath)/md407-ram.ux;-L\$(ARM_V6LIB) -L\$(ARM_GCC_V6LIB);-nostartfiles;
Libraries Search Path	<PathToLibrary>
Libraries	gcc;c_nano;md407

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:



Options

Use with global settings	append
Linker Options	-T\$(ProjectPath)/md407-ram.ux;-L\$(ARM_V6LIB) -L\$(ARM_GCC_V6LIB);-nostartfiles;
Libraries Search Path	<PathToLibrary>
Libraries	gcc;c_nano;md407

Demonstration: Användning av biblioteket

