Runtime environment and libraries

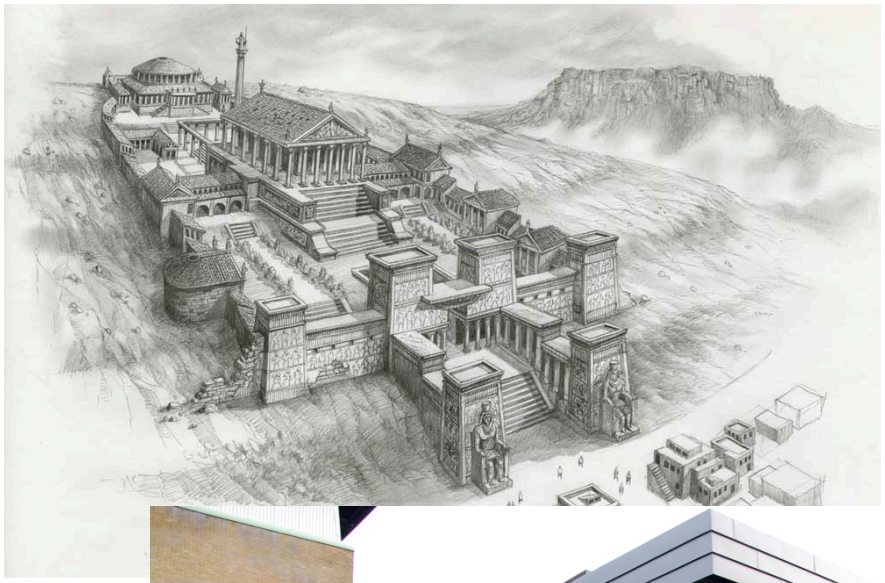
Ur innehållet:

- Olika typer av bibliotek
- Kompilatorbibliotek
- C-bibliotek
- Run-time miljö
- Så skapar du ett nytt bibliotek

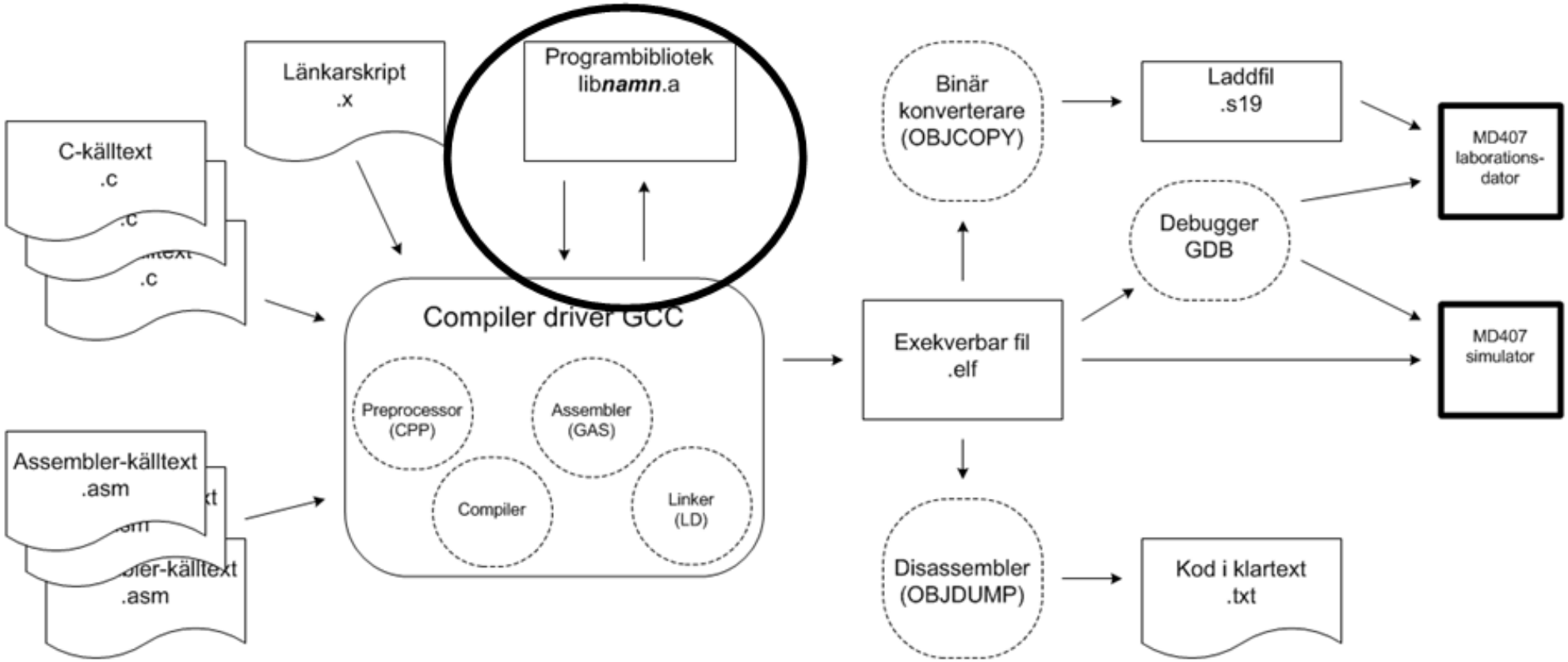
Läsanvisningar:

- Arbetsbok kapitel 8

Libraries…



Libraries


Länkarskript
.x
Programbibliotek
lib*namn*.a
C-källtext
.c
.c
.c
Assembler-källtext
.asm
.asm
Compiler driver GCC
Preprocessor
(CPP)
Assembler
(GAS)
Compiler
Linker
(LD)
Exekverbar fil
.elf
Binär
konverterare
(OBJCOPY)
Laddfil
.s19
Debugger
GDB
MD407
laborations-
dator
MD407
simulator
Disassembler
(OBJDUMP)
Kod i klartext
.txt

Libraries – various types

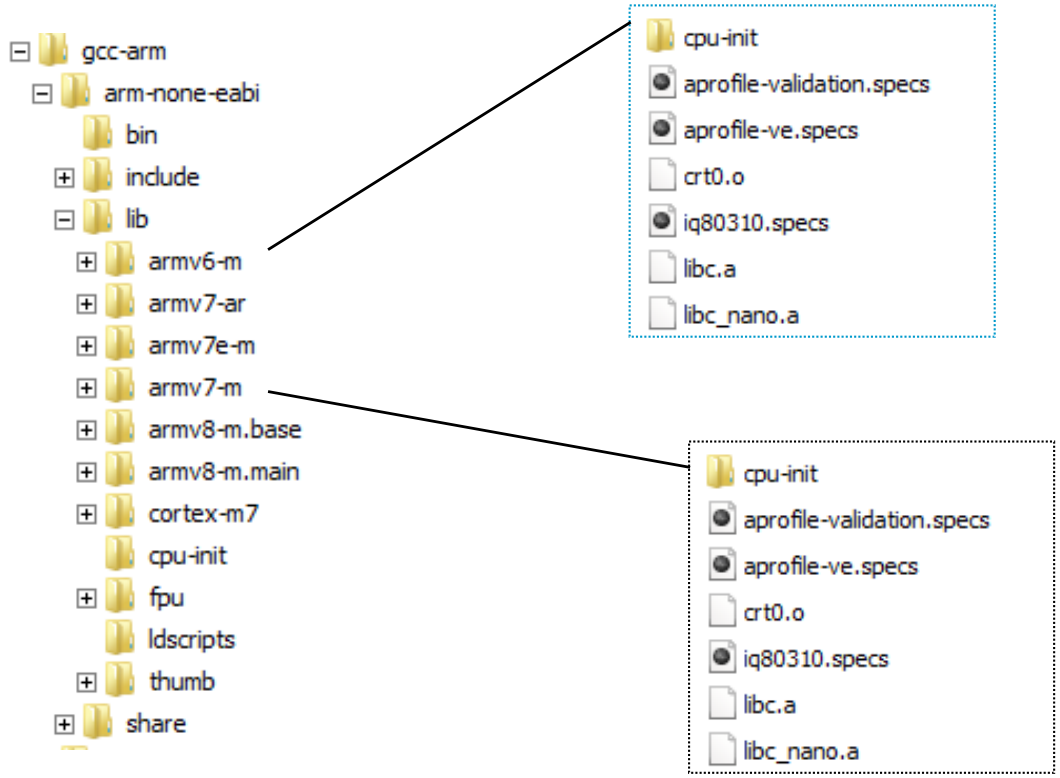
Det finns en rad olika typer av färdiga programbibliotek. De flesta prograbibliotek förutsätter vanligtvis att det finns flera, i bland åtskilliga, underliggande bibliotek. Typexempel på underliggande bibliotek är så kallade "kompatibilitetslager" där en anpassning sker till något specifikt operativsystem och någon speciell maskinvara.

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

libc.a

SDL: Simple DirectMedia Library
libsdl.a
Mathematics:
#include <math.h>
libm.a

”Multilib” – various configurations



Det finns en konfiguration (uppsättning programbibliotek och startfiler) för *varje* arkitektur.

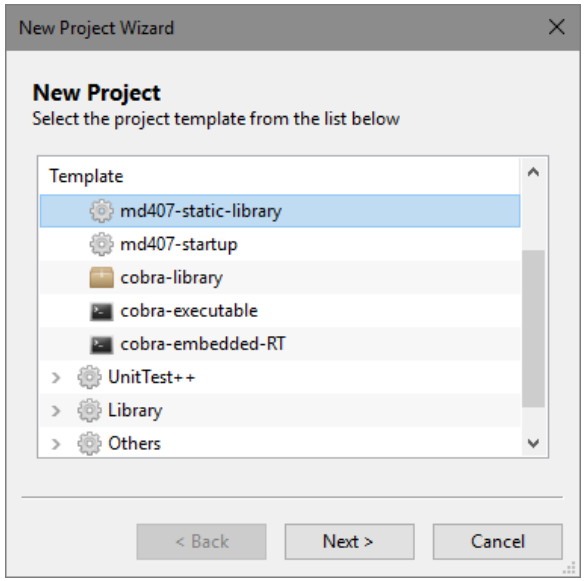
På detta sätt kan optimal kod länkas till applikationen.

I listan av bibliotek utelämnar man (av konvention) delar av namnet, dvs:
`lib`**`gcc`**`.a` skrivs `gcc`
`lib`**`c_nano`**`.a` skrivs `c_nano`

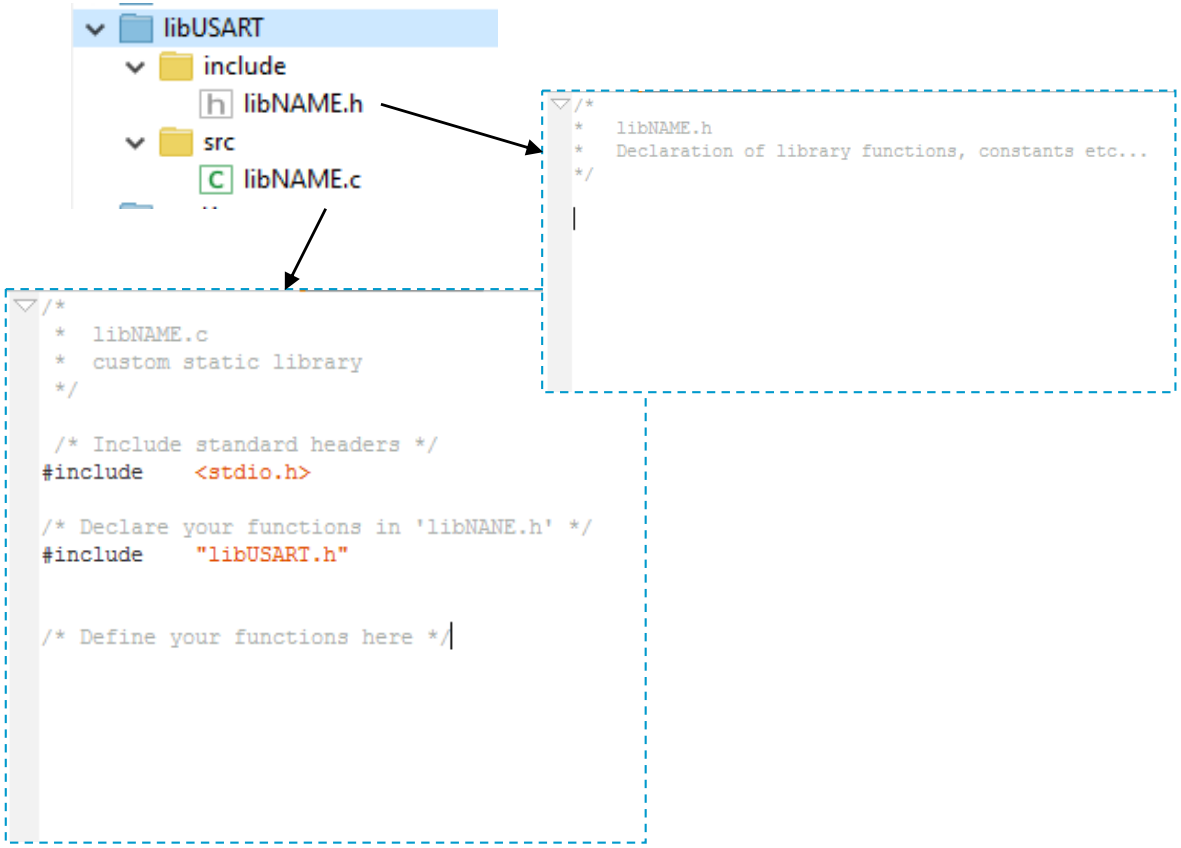
Linker Options	-nostartfiles;-T$(ProjectPath)/md407-ram.x
Libraries Search Path	$(CodeLiteDir)/tools/gcc-arm/arm-none-eabi/lib/armv6-m;$(CodeLiteDir)/tools/gcc-arm/lib/gcc/arm-none-eabi/5.4.1/armv6-m
Libraries	gcc;c_nano

EXAMPLE: Library with USART-routines

Skapa ett nytt projekt: **`libUSART`**, baserat på mallen *md407-static-library*.



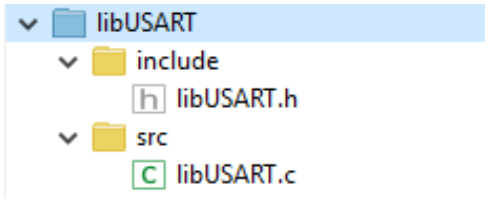
Två "tomma" filer kopieras från mallen:



EXAMPLE: Library with USART-routines

Från projektmallen kopieras två filer, döp om dessa:

- *En header-fil för deklarationer, libNAME.h -> libUSART.h*
- *En c-fil för programkod, libNAME.c -> libUSART.c*
- *Kopiera USART-funktioner från tidigare projekt*




```c
/*
*   libUSART.h
*   Declaration of library functions, constants etc...
*/

typedef struct tag_usart
{
    volatile unsigned short sr;
    volatile unsigned short Unused0;
    volatile unsigned short dr;
    volatile unsigned short Unused1;
    volatile unsigned short brr;
    volatile unsigned short Unused2;
    volatile unsigned short cr1;
    volatile unsigned short Unused3;
    volatile unsigned short cr2;
    volatile unsigned short Unused4;
    volatile unsigned short cr3;
    volatile unsigned short Unused5;
    volatile unsigned short gtpr;
} USART;

#define UE      (1<<13)
#define TE      (1<<3)
#define RE      (1<<2)
#define TXE     (1<<7)
#define RXNE    (1<<5)

#define USART1  ((USART *) 0x40011000)

void    _init( void );
void    _outchar( char c );
char    _tstchar( void );
char    _inchar( void );
```

```c
/*
 *  libUSART.c
 *  custom static library
 */

/* Declare your functions in 'libNANE.h' */
#include    "libUSART.h"

/* Define your functions here */
void    _init( void )
{
    USART1->brr = 0x2D9;
    USART1->cr3 = 0;
    USART1->cr2 = 0;
    USART1->cr1 = UE | TE | RE;
}

void    _outchar( char c )
{
    while (( USART1->sr & TXE )==0);
    USART1->dr = (unsigned short) c;
    if( c == '\n')
            _outchar('\r');
}

char    _tstchar( void )
{
    if( (USART1->sr & RXNE )==0)
        return 0;
    return (char) USART1->dr;
}

char    _inchar( void )
{
    char c;
    while ( (c=_tstchar() )==0);
    return c;
}
```

EXAMPLE: Library with USART-routines

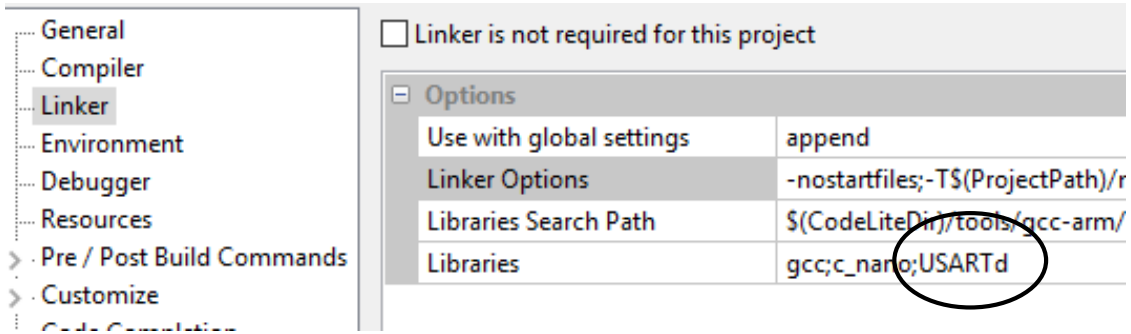
Installera biblioteket genom att kopiera filerna till deras "standardplatser"...

- `libUSART.h` kopieras till:
  `$(CodeLiteDir)/tools/gcc-arm/arm-none-eabi/include/libUSART.h`

- `Debug/libUSARTd.a` och `Release/libUSART.a` kopieras till:
  `$(CodeLiteDir)/tools/gcc-arm/arm-none-eabi/lib/armv6-m`

Skapa ett testprojekt *libUSARTtest* för att testa biblioteket

- Lägg till `USARTd` i listan av bibliotek




```c
/*
 *  libUSARTtest.c
 *
 */

 #include   <libUSART.h>

void startup(void) __attribute__((naked)) __attribute__(

void startup ( void )
{
asm volatile(
    " LDR R0,=0x2001C000\n"     /* set stack */
    " MOV SP,R0\n"
    " BL _init\n"               /* init UART library */
    " BL main\n"                /* call main */
    ".L1: B .L1\n"              /* never return */
    ) ;
}

void main(void)
{
    unsigned char c;
    while(1){
        c = _inchar();
        _outchar( c );
    }
 }
```

Standard C library

ket C och

```
#include <stdio.h>
#include <float.h>
int main () {
  printf("The maximum value of float = %.10e\n", FLT_MAX);
  print
  prin
}
```

```
#include <stdio.h>
#include <math.h>
int main () {
  double y, x = 2.5;
  y = sqrt( x );
  printf("The square root of %f is %f\n", x, y);
}
```

```
#include <stdio.h>
#include <string.h>
int main () {
  char *s1 = ”Hello”;
  char *s2 = ”World”;
  if( strcmp( s1, s2 ) == 0 )
    printf(”Strings s1 and s2 are the same!\n");
}
```

<st	minnesha
<string.h>	Deklarerar funktioner för strånghantering.
<time.h>	Deklarerar funktioner för datum och tid.

Use of the standard C-library

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

main.c

```
#include <stdio.h>

...
...
printf( "Hello World!");
...
```

arm-none-eabi/include/stdio.h

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

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

```
10001010011100011001010
printf:
00101100001010001001001001010
1000101001110001100101.....
```

Startup...

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

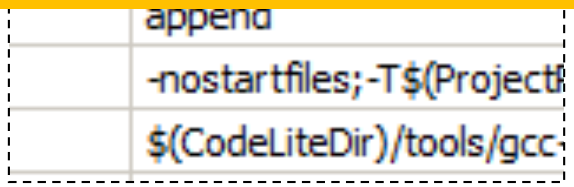
Startup-sekvensen blir system-beroende, ex: vilken microcontroller som används

```
void startup ( void )
{
asm volatile(
 " LDR R0,=0x2001C000\n"
 " MOV SP,R0\n"
 " BL _usart_init\n"
 " BL _crt_init\n"
 " BL main\n"
 " BL _deinit\n"
 ".L1: B .L1\n"
 );
}
```

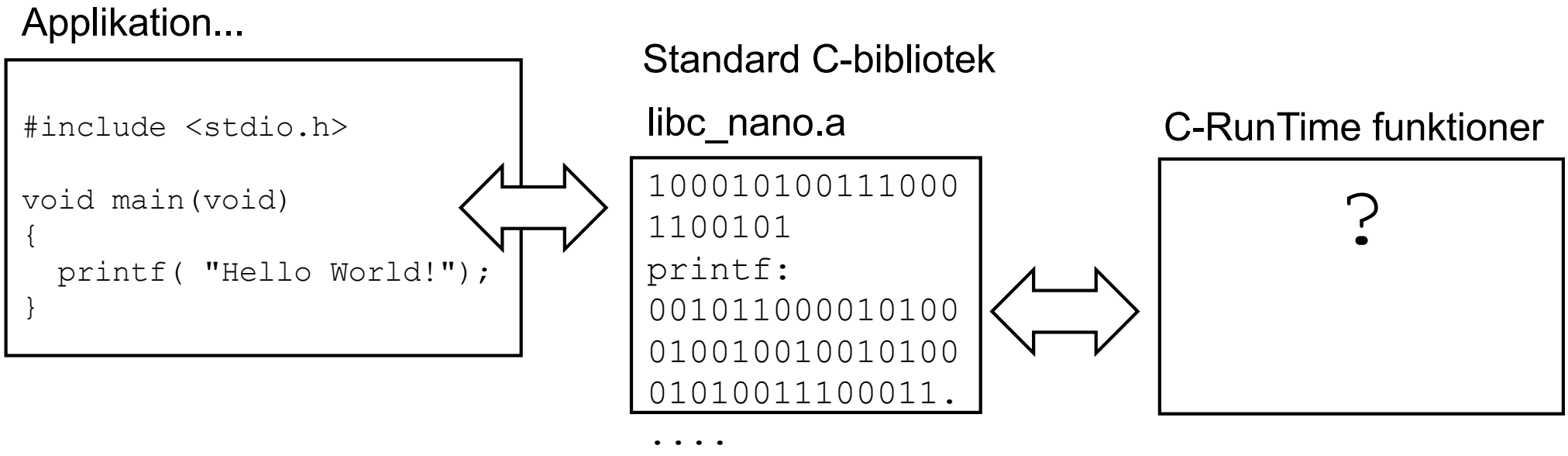
main.c

```
#include <stdio.h>

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

För att *undvika* länkning med standard `crt0.o` ger man länkarflaggan `-nostartfiles`



"C-nano" – lightweight implementation



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:

`_sbrk, _write, _close, _lseek, _read, _fstat, _isatty`

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

"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` definieras 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" – lightweight implementation for MD407

`_open`, öppna en fil för läsning och/eller skrivning.
Vår implementering stödjer endast terminalanslutning (stdin, stdout, stderr). Dessa är alltid öppna av konvention, 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 stdin, stdout och stderr 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 stdin, stdout och stderr.

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

"C-run time" – lightweight 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) {
  int todo;
  for (todo = 0; todo < len; todo++) {
    _outchar( *ptr++ );
  }
  return len;
}
```

`_read`, läs från en öppen fil.

```
int _read(int file, char *ptr, int len) {
  int todo;
  if(len == 0) return 0;
  for(todo = 1; todo < len; todo++) {
    *ptr++ = _inchar();
  }
  return todo;
}
```

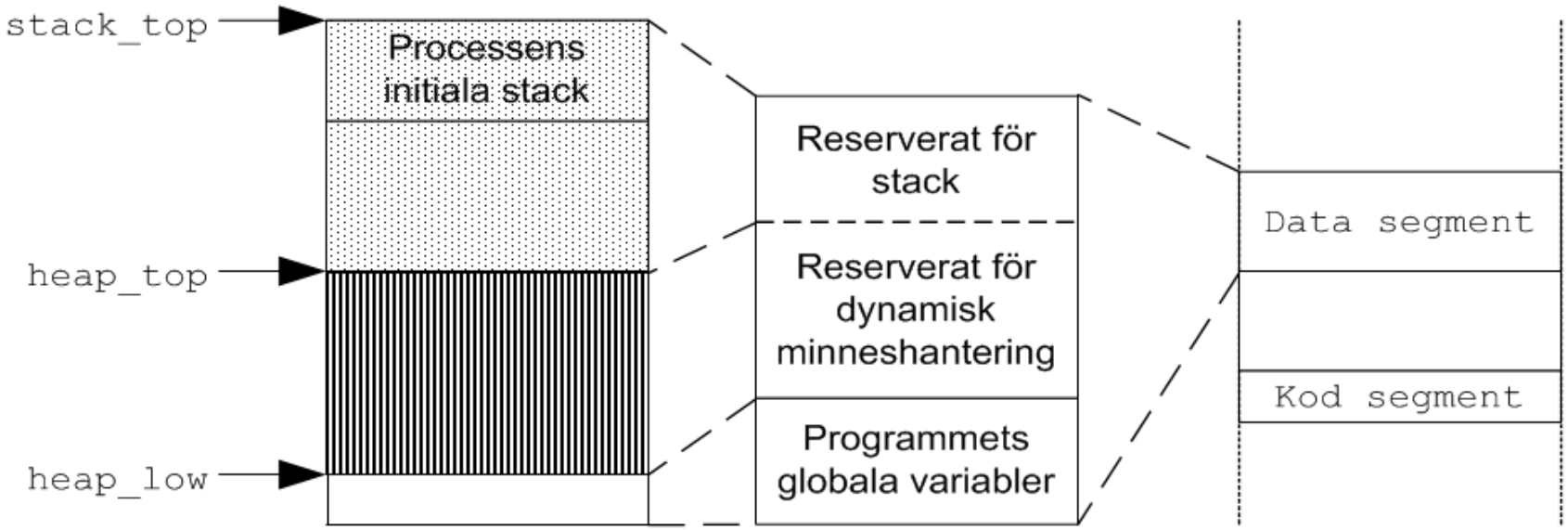
Dynamic memory management, malloc/free

C-biblioteket tillhandahåller rutiner som `malloc` och `free` för dynamisk minneshantering.

Runtime biblioteket måste då 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

```
...
   *(.rodata
   *(.rodata.*)
   . = ALIGN(4);
} >RAM
   . = ALIGN(8);
   heap_low = .; /* for _sbrk */
   . = . + 0x400; /* 1 kB heap */
   heap_top = .; /* for _sbrk */
   . = . + 0x400; /* 1 kB stack */
   stack_top = .; /* for startup */
...
```

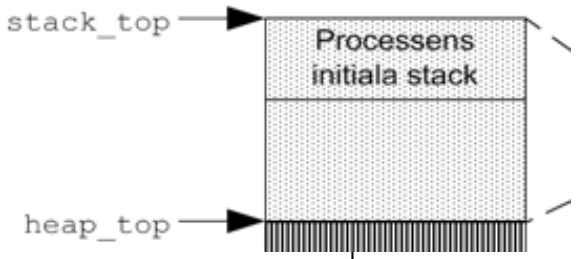


Implementation of _sbrk

```
#include       <errno.h>
extern char heap_low; /* Defined by linker */
extern char heap_top; /* Defined by linker */
char *heap_end;

char * _sbrk(int incr) {
  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 *)-1;
  }
  heap_end += incr;
  return (char *) prev_heap_end;
}
```

Länkaren skapar de symboler vi behöver för att administrera minnet

```
...
  *(.rodata
  *(.rodata.*)
  . = ALIGN(4);
} >RAM
  . = ALIGN(8);
  heap_low = .; /* for _sbrk */
  . = . + 0x400; /* 1 kB heap */
  heap_top = .; /* for _sbrk */
  . = . + 0x400; /* 1 kB stack */
  stack_top = .; /* for startup */
...
```



Initialization of the r

BSS (from Block Started by Symbol)
The BSS segment typically includes all uninitialized objects (both variables and constants) declared at file scope (i.e., outside any function) as well as uninitialized static local variables

```
void crt_init() {
extern char _sbss; /* Defined by linker */
extern char _ebss; /* Defined by linker */
  char *s;
  s = &_sbss;
  while( s < &_ebss )
    *s++ = 0;
    s = &heap_low;
    while( s < &heap_top )
      *s++ = 0;
    heap_end = 0;
  /* stäng av buffring */
  setvbuf(stdin, NULL, _IONBF, 0);
  setvbuf(stdout, NULL, _IONBF, 0);
  setvbuf(stderr, NULL, _IONBF, 0);
}
```

Initialized statically-allocated variables

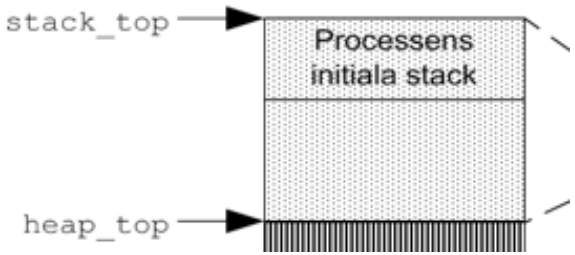
Initialized heap space

NBF = No Buffer

Även här låter vi länkaren skapa de symboler vi behöver för att initiera runtime-funktionerna

```
...
 .text :
{
   . = ALIGN(4);
   *(.start_section) /* startup code */
 *(.text)            /* remaining code */
 *(.text.*)
 _sbss = .; /* start bss */
 *(.bss)  /* unintialised data */
 *(COMMON)
 _ebss = .; /* end bss */
 *(.data) /* initialised data */
 *(.data.*)
 *(.rodata) /* read-only data (constants) */
 *(.rodata.*)
 . = ALIGN(4);
} >RAM
```

Implementation of a modified startup



Länkaren skapar de symboler vi behöver för att administrera minnet

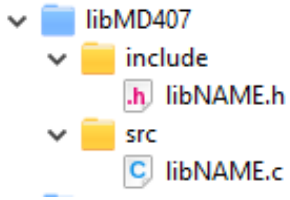
```
...
   *(.rodata
   *(.rodata.*)
   . = ALIGN(4);
} >RAM
   . = ALIGN(8);
   heap_low = .; /* for _sbrk */
   . = . + 0x400; /* 1 kB heap */
   heap_top = .; /* for _sbrk */
   . = . + 0x400; /* 1 kB stack */
   stack_top = .; /* for startup */
...
```

```
void startup ( void )
{
asm volatile(
        " LDR R0,=stack_top\n"          /* set stack */
        " MOV SP,R0\n"
        " BL  crt_init\n"               /* init C-runtime library */
        " BL  main\n"                   /* call main */
        ".L1: B .L1\n"                  /* never return */
        ) ;
}
```

Create runtime library for MD407

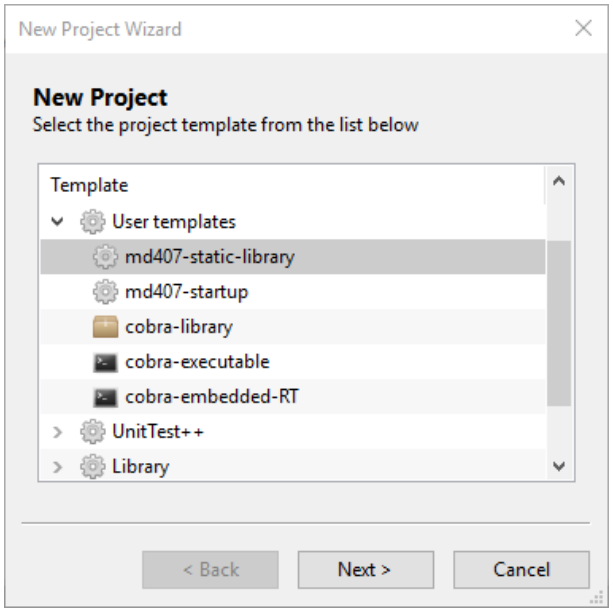
EXEMPEL

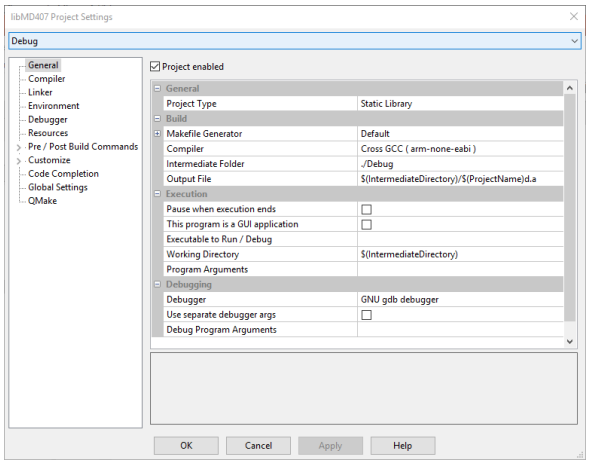
För att slippa kompilera vårt lilla runtime-bibliotek tillsammans med applikationen skapar vi i stället ett programbibliotek `libMD407.a`,
(hämta mallen från kursens hemsida och installera...)
eller skapa en egen mall..




```
/*
*   libNAME.h
*   Declaration of library functions, constants etc...
*/
```

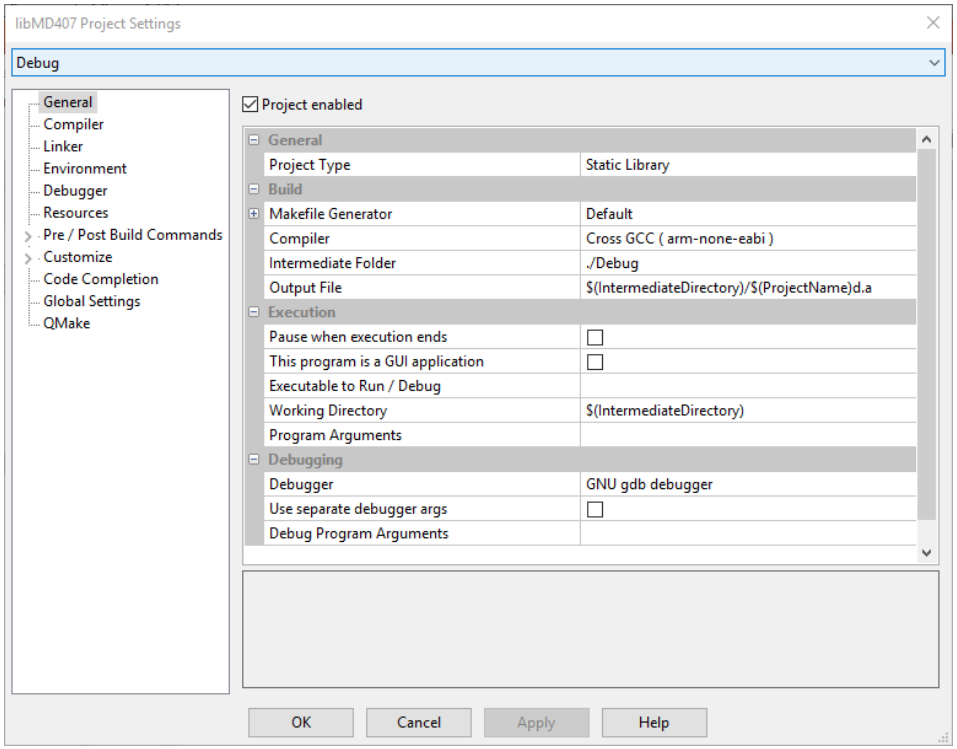
```
/*
 *  libNAME.c
 *  custom static library
 */

 /* Include standard headers */
#include    <stdio.h>

/* Declare your functions in 'libNANE.h' */
#include    "libNAME.h"


/* Define your functions here */
```



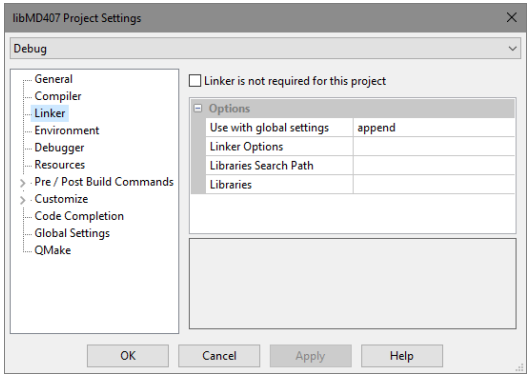


md407-static-library, settings...

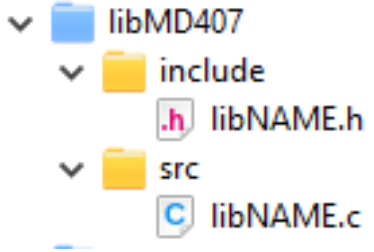
Börja med att skapa ett vanligt projekt, ändra till 'Static library'



I stället för länkaren används arkivhanteraren (AR) här



Ta bort 'resources' (behövs inte) men lägg till 'include' och skapa filerna:



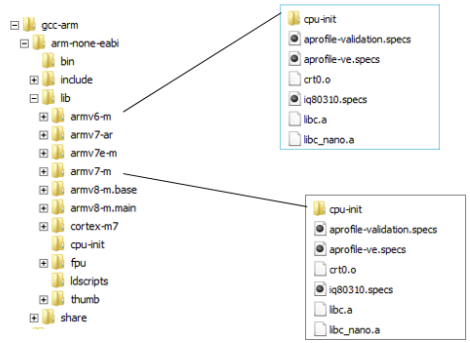
md407-static-library, source files

```
/*
 * 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() {
           char *s;
...
```

```
/*
* libMD407.h
* Declaration of library functions, constants etc...
*/
#include  <stdio.h>
#include  <errno.h>
#include  <sys/stat.h>
/* Type definitions */
typedef struct tag_usart
{
          volatile unsigned short sr;
etc...
/* Constants */
#define USART1  ((USART *) 0x40011000)
/* Constant defined by linker */
extern char heap_low;
extern char heap_top;
extern char _sbss;
extern char _ebss;
/* Library defined functions */
void      _outchar( char c );
char      _tstchar( void );
char      _inchar( void );

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

md407-static-library, installation and use

”Multilib” – olika konfigurationer



Det finns en uppsättning programbibliotek och startfiler för *varje* konfiguration.

På detta sätt kan optimal kod länkas till applikationen.

I listan av bibliotek utelämnar man (av konvention) delar av namnet, dvs:
`libgcc.a` skrivs `gcc`
`libc_nano.a` skrivs `c_nano`

Linker Options	-nostartfiles;-T${ProjectPath}/md407-ram.x
Libraries Search Path	$(CodeLiteDir)/tools/gcc-arm/arm-none-eabi/lib/armv6-m;$(CodeLiteDir)/tools/gcc-arm/lib/gcc/arm-none-eabi/5.4.1/armv6-m
Libraries	gcc;c_nano

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:

	$(CodeLiteDir)/tools/gcc-arm
	gcc;c_nano;MD407d

Du måste också använda den utökade script-filen för länkaren, för din applikation:

```
SECTIONS
{
  .text :
  {
     . = ALIGN(4);
     *(.start_section)
     ...
     *(.rodata
     . = ALIGN(4);
   } >RAM
   . = ALIGN(8);
  heap_low = .; /* for _sbrk */
  . = . + 0x400; /* 1 kB heap  */
  heap_top = .; /* for _sbrk */
  . = . + 0x400; /* 1 kB stack  */
  stack_top = .; /* for startup */
}
```

Compiler library

test.c

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

test.s (arm-**v6m**)

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

test.s (arm-**v7m**)

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

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

```
10001010011100011001O1
__aeabi_idiv:
00101100001010001001001O010
1000101001110001 1.....
```

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.

Float

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:

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

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...

Float

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]
```

VCVT (Vector Convert) from fixed point or integer to float
F32.S32 - signed integer to floating-point