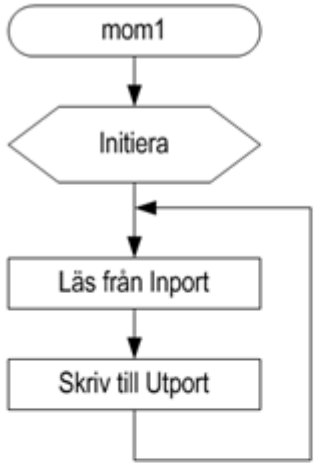
Assemblerprogrammering för ARM – del 1

Ur innehållet:

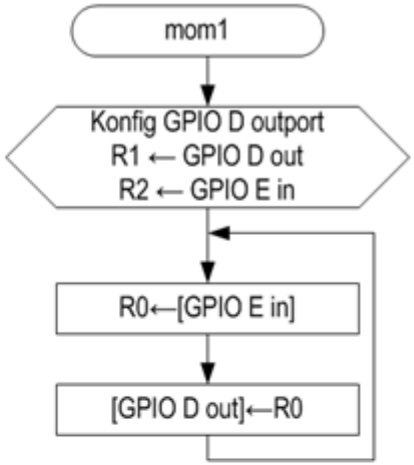
- Assemblerspråk
- Ordlängder och datatyper
  - Variabeldeklarationer
- Programkonstruktioner
  - Tilldelningar
  - Uttrycksevaluering
  - Ovillkorliga programflöden
- Läsanvisningar:
  - Arbetsbok kap 1 och 2
  - Quick-guide, instruktionslistan

Assembler-programmets struktur; exempel

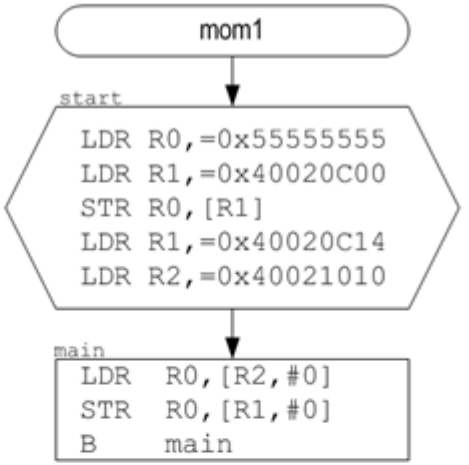
Specifikation



Registerallokering



Implementering



@			alla tecken efter @ betraktas som kommentarer
@	mom1.asm		
@			
start:	LDR	R0,=0x55555555	sätt upp port D som utport
	LDR	R1,=0x40020C00	
	STR	R0,[R1]	
	LDR	R1,=0x40020C14	@ utport skapar en pekare till port D i R1
	LDR	R2,=0x40021010	@ inport skapar en pekare till port E i R2
main:	LDR	R0,[R2]	laddar 32 bitar från port E i R0
	STR	R0,[R1]	skriver innehållet i R0 till port E
	B	main	absolut programflödesändring (B=*branch*)

Adresseringssätt syntetiseras i själva verket av assemblatorn, exempelvis: `LDR R0,=symbol` ger symbols värde i `R0`

Vi testar i simulatorn...

Assemblerspråkets element

ALLA textsträngar är ”context”-beroende

”Mnemonic”, ett ord som om det förekommer i instruktionsfältet tolkas som en assemblerinstruktion ur processorns instruktionsuppsättning. Mot varje sådan mnemonic svarar som regel EN maskininstruktion.

”Assemblerdirektiv”, ett direktiv till assemblatorn.

Symboler, textsträng. Ska bara förekomma i symbol- eller operand- fälten, I symbolfältet ska dessa alltid avslutas med **:** (kolon)

Direktiv och mnemonics är inte ”reserverade” ord i vanlig bemärkelse utan kan till exempel också användas som symbolnamn

Ordlängder och heltalstyper, ANSI C

Typ	Förklaring
`char`	Minsta adresserbara enhet som kan rymma en basal teckenuppsättning. Det är dock en heltalstyp som kan vara signed eller unsigned, detta är implementationsberoende.
`signed char`	Tolkas som heltal med tecken. Måste **minst** kunna representera talområdet [−127, +127], dvs. minst 8 bitar.
`unsigned char`	Tolkas som heltal utan tecken. Måste **minst** kunna representera talområdet [0, 255], dvs. minst 8 bitar.
`short` **`short int`** **`signed short`** **`signed short int`**	Kort heltalstyp med tecken. Måste **minst** kunna representera talområdet [−32767, +32767], dvs. minst 16 bitar. Observera att talområdet [−32768, +32767] som fås vid 2-komplementsrepresentation, också är tillåtet.
`unsigned short` **`unsigned short int`**	Kort heltalstyp utan tecken. Måste **minst** kunna representera talområdet [0, +65535], dvs. minst 16 bitar.
`long` **`long int`** **`signed long`** **`signed long int`**	Lång heltalstyp med tecken. Måste **minst** kunna representera talområdet [−2147483647, +2147483647], dvs. minst 32 bitar.
`unsigned long` **`unsigned long int`**	Lång heltalstyp utan tecken. Måste **minst** kunna representera talområdet [0, +4294967295], dvs. minst 32 bitar.
`int` **`signed`** **`signed int`**	Implementationsbestämd heltalstyp med tecken. Måste **minst** kunna representera talområdet [−32767, +32767], dvs. minst 16 bitar. Observera att talområdet [−32768, +32767] som fås vid 2-komplementsrepresentation, också är tillåtet. Observera speciellt att `int` kan vara synonym med `short` eller `long`.
`unsigned` **`unsigned int`**	Typen `int` utan tecken.

Rättningsvillkor ("alignment")

Av prestandaskäl har vi restriktioner på var olika datatyper kan placeras i minnet:

int (32-bitar, word) får bara finnas på en adress jämnt delbar med 4, s.k "word alignment")

short (16-bitar, halfword) får bara finnas på en adress jämnt delbar med 2, s.k "halfword alignment")

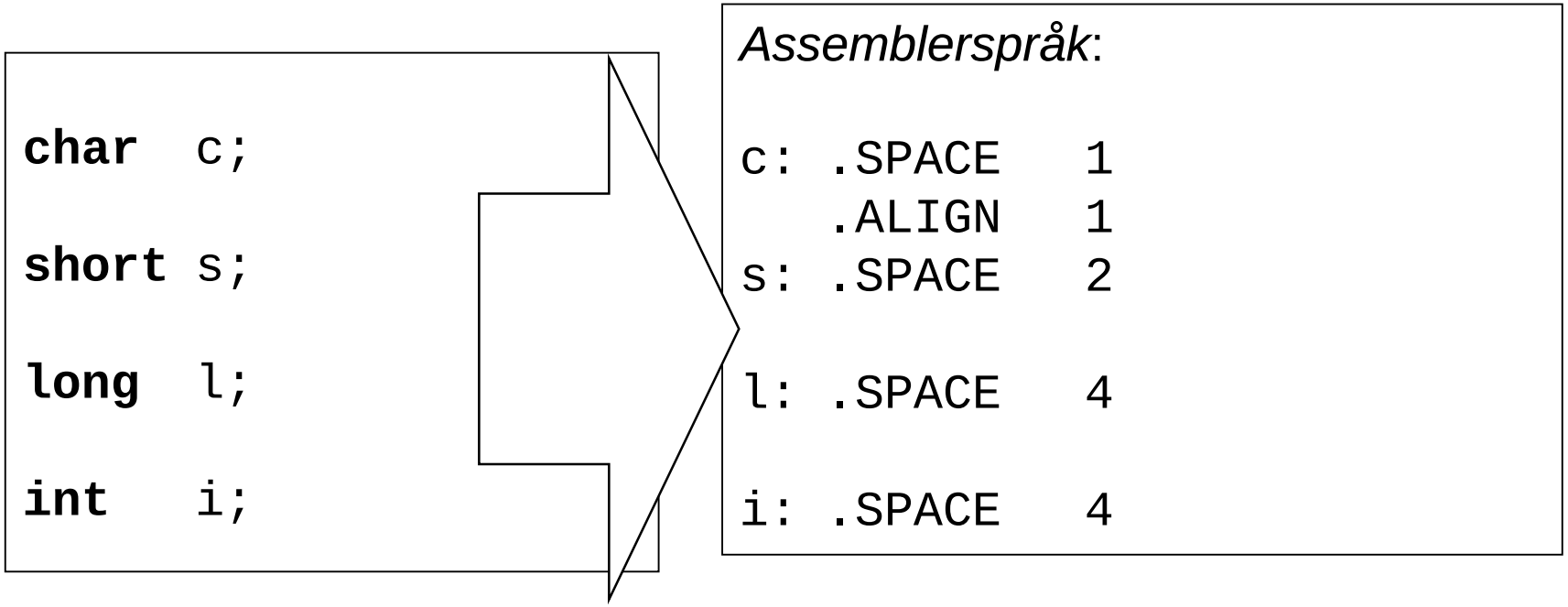
char (8 bitar) kan finnas på såväl udda som jämn adress

Assemblerdirektiv:

```
.align   {n}  @ bytejustera efterföljande kod
              @ (n=4 default)
```

Ordlängder och heltalstyper

```
char      c;    /* 8-bitars datatyp, storlek byte */
short     s;    /* 16-bitars datatyp, storlek halfword */
long      l;    /* 32-bitars datatyp, storlek word */
int       i;    /* 32-bitars datatyp, storlek word */
```



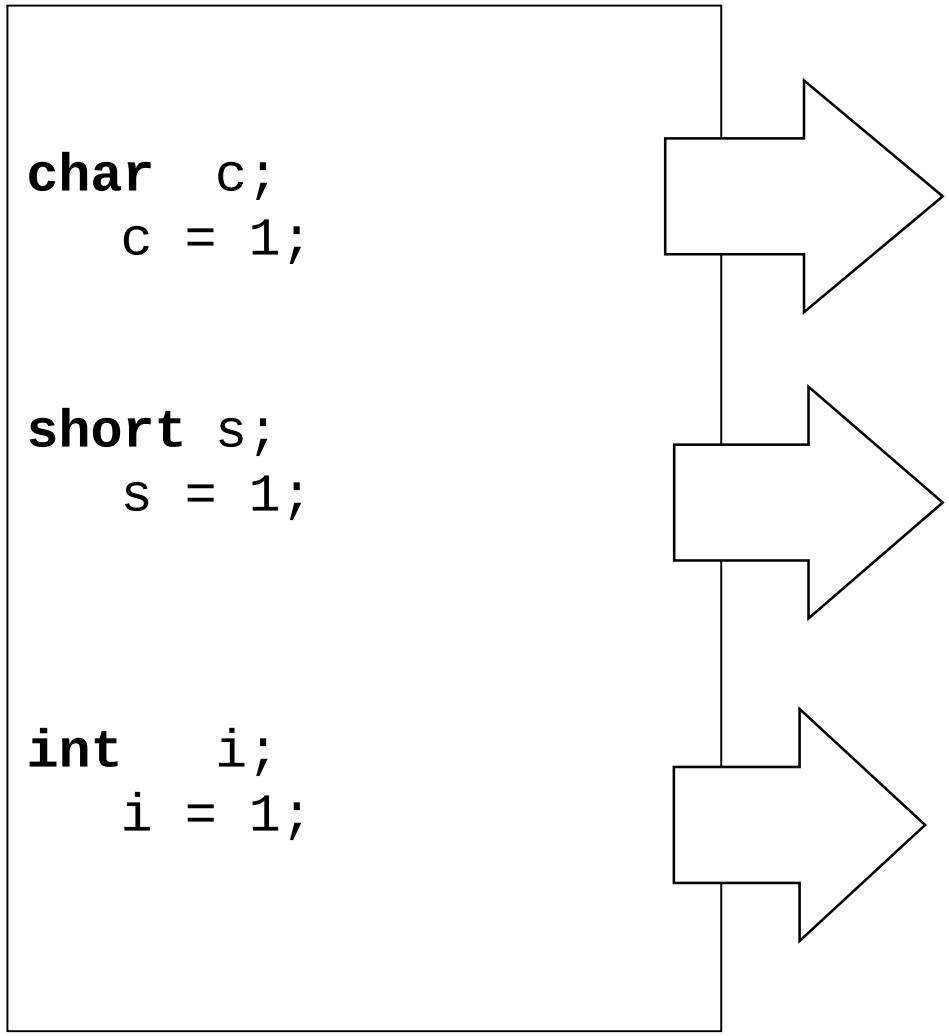
Instruktioner för tilldelningar

Instruktion	Betydelse	Datatyp
LDRB	Load byte	char
LDRH	Load halfword	short
LDR	Load word	int
MOV	Move	(int & 0xFF)
STRB	Store byte	char
STRH	Store halfword	short
STR	Store word	int

Adresseringssätt

Namn			
Register direct	Rx	MOV R0,R1	R0←R1
Direct	Symbol	LDR R0,symbol	R0←M(symbol)
Immediate	#const	MOV R0,#0x15	R0←0x15
Register indirect	[Rx]	LDR R0,[R1]	R0←M(R1)

Tilldelningar



Assemblerspråk:

```
c:      .SPACE    1
MOV     R0,#1
LDR     R7,=c
STRB    R0,[R7]
.....
s:      .SPACE    2
MOV     R0,#1
LDR     R7,=s
STRH    R0,[R7]
.....
i:      .SPACE 4
MOV     R0,#1
LDR     R7,=i
STR     R0,[R7]
.....
```

EXEMPEL

Antag följande deklarationer på ”toppnivå”:

```
unsigned char    a;
signed char      c;
unsigned short   b;
signed short     d;
```

Koda följande tilldelningssatser i assembler:

```
a = (unsigned char) b;
c = (signed char) d;
b = (unsigned short) a;
d = (signed short) c;
```

Vi löser på tavlan...

Uttrycksevaluering

DEST = (type of DEST) (uttryck); "värde uttryck"

EXEMPEL:

```
int a,b,c;
a = b+c;
```

(uttryck); "sanningsuttryck", dvs ==0 eller != 0

EXEMPEL:

```
(a+1);      /* falskt om a==-1 */
(a <= b);   /* falskt om a>b */
```

Instruktioner för unära operationer

C-operator	Betydelse	Datatyp	Instruktion
~	bitvis not	signed/unsigned	MVN
-	negation	signed/unsigned	RSB

DEST = *operation* (type of DEST) OPERAND;

EXEMPEL: Antag följande deklarationer på ”toppnivå”:

```
char    a,c;
```

Koda följande tilldelningssatser i assembler:

```
a = ~c;
a = -c;
```

Vi löser på tavlan...

Instruktioner för binära operationer

C-operator	Betydelse	Datatyp	Instruktion
+	addition	signed/unsigned	ADD
-	subtraktion	signed/unsigned	SUB
*	multiplikation	signed/unsigned	MUL
/	division	signed/unsigned	DIV
&	bitoperation AND	signed/unsigned	AND
|	bitoperation OR	signed/unsigned	OR
^	bitoperation EOR	signed/unsigned	EOR
<<	skift vänster	signed/unsigned	LSL
>>	skift höger	signed unsigned	ASR LSR

Kodgenerering, binära operationer

DEST = (type of DEST) OPERAND1 *operation* (type of DEST) OPERAND2;

Exempel för 16 bitars ord:

```
LDRH    R1,OPERAND1
   (ev. teckenutvidga)
LDRH    R2,OPERAND2
   (ev. teckenutvidga)
operation R0,R1,R2
LDR     R7,=DEST
STR     R0,[R7]
```

På tavlan..

Addition av 32-bitars tal

```
int    ia,ib,ic;
 ...
ia = ib + ic;
```

Assemblerspråk:

```
ia:   .SPACE   4
ib:   .SPACE   4
ic:   .SPACE   4
...
   LDR   R1,ia        @ källoperand 1
   LDR   R2,ib        @ källoperand 2
   LDR   R7,=ia       @ adress till destinationsoperand

   ADD   R0,R1,R2     @ operation
   STR   R0,[R7]      @ tilldelning
```

På tavlan..

Addition av 16-bitars tal

Eftersom alla instruktioner opererar på 32-bitar måste ingående operander först anpassas:

```
short int ssa,ssb,ssc;
   ...
 sa = sb + sc;
```

```
sa:.SPACE    2
sb:.SPACE    2
sc:.SPACE    2
...
    LDR    R1,=sb
    LDRH   R1,[R1]
    SXTH   R1,R1
    LDR    R2,=sc
    LDRH   R2,[R2]
    SXTH   R2,R2
    ADD    R0,R1,R2
    LDR    R7,=sa
    STRH   R0,[R7]
```

```
unsigned short int  usa,usb,usc;
   ...
 usa = usb + usc;
```

```
usa:   .SPACE    2
usb:   .SPACE    2
usc:   .SPACE    2
...
    LDR    R1,=usb
    LDRH   R1,[R1]
    UXTH   R1,R1      @ teckenutvidga
    LDR    R2,=usc
    LDRH   R2,[R2]
    UXTH   R2,R2      @ teckenutvidga
    ADD    R0,R1,R2
    LDR    R7,=usa
    STRH   R0,[R7]
```

På tavlan..

Addition av 8-bitars tal

Andra instruktioner för teckenutviddgning, i övrigt samma kod:

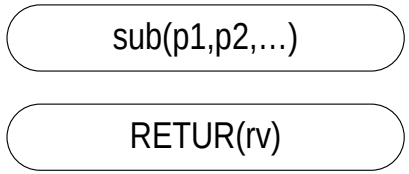
```
char   ca,cb,cc;
   ...
 ca = cb + cc;
```

```
ca:.SPACE    1
cb:.SPACE    1
cc:.SPACE    1
...
    LDR    R1,=cb
    LDRB   R1,[R1]
    SXTB   R1,R1
    LDR    R2,=cc
    LDRB   R2,[R2]
    SXTB   R2,R2
    ADD R0,R1,R2
    LDR R7,=ca
    STRB   R0,[R7]
```

```
 unsigned char    uca,ucb,ucc;
   ...
  uca = ucb + ucc;
```

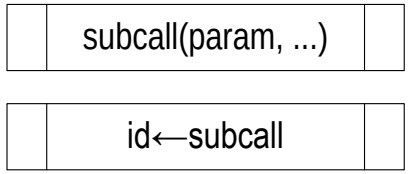
```
uca:   .SPACE    1
ucb:   .SPACE    1
ucc:   .SPACE    1
...
    LDR    R1,=ucb
    LDRB   R1,[R1]
    UXTB   R1,R1      @ teckenutvidga
    LDR    R2,=ucc
    LDRB   R2,[R2]
    UXTB   R2,R2      @ teckenutvidga
    ADD R0,R1,R2
    LDR R7,=uca
    STRB   R0,[R7]
```

Flödesdiagram för programstrukturer



Inträde i och utträde ur subrutiner.

Inträde, typiskt med subrutinens namn och symbolisk representation av eventuella parametrar då sådana finns.
Utträde, (”RETUR”) typiskt med angivande av returnvärde (rv) om sådant finns.

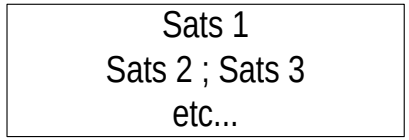


Subrutinanrop.

Med parametrar, symbolisk representation av eventuella aktuella parametrar som skickas med subrutinanropet.
Med returvärde, tilldelningsoperator placeras framför den anropade subrutinen.

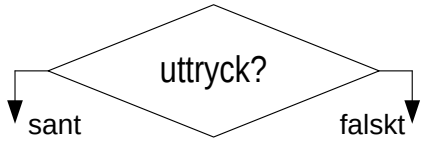
Engångsinitieringar.

Placeras typiskt i omedelbar anslutning till en inträdessymbol



Exekveringsblock.

En eller flera satser som ordnats och exekveras sekvensiellt.



Villkorsblock.

Ett uttryck testas, utfallet kan vara sant eller falskt och exekveringsväg väljs därefter.