Korsutveckling för ARM/Thumb – del 2 (sammanfattning)

Ur innehållet

Programflöde
Subrutiner, parametrar och returvärden
Tillfälliga (lokala) variabler

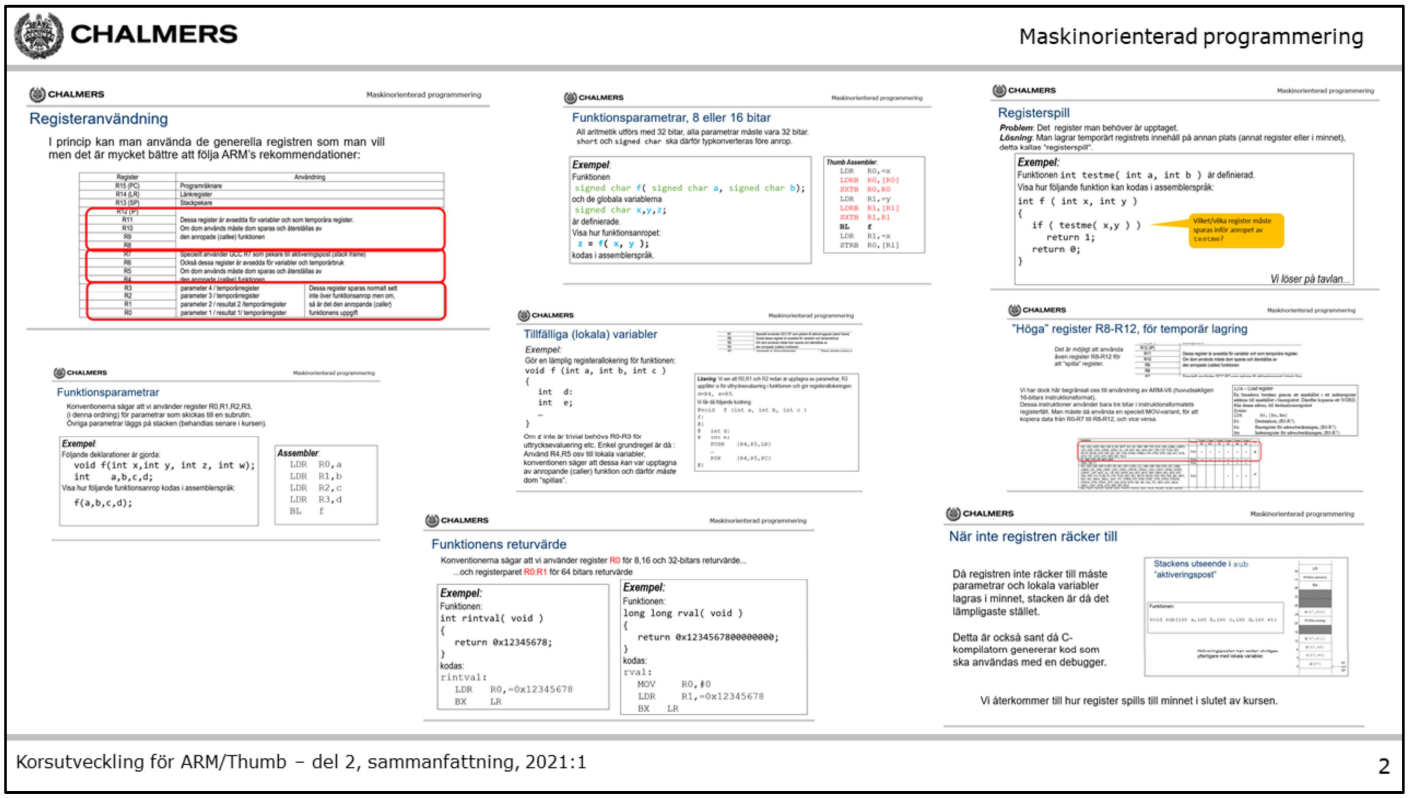
Läsanvisningar:

Arbetsbok kap 2
Quick-guide, instruktionslistan

Målsättningar:

Koda och testa enkla C-funktioner.
Utforma och testa assemblerkod med MD407-simulator (ETERM8)

Denna lektion ger en introduktion till hur processorns register bör användas.
Vi tittar också på kodning med instruktioner för villkorliga programflödeskontroll



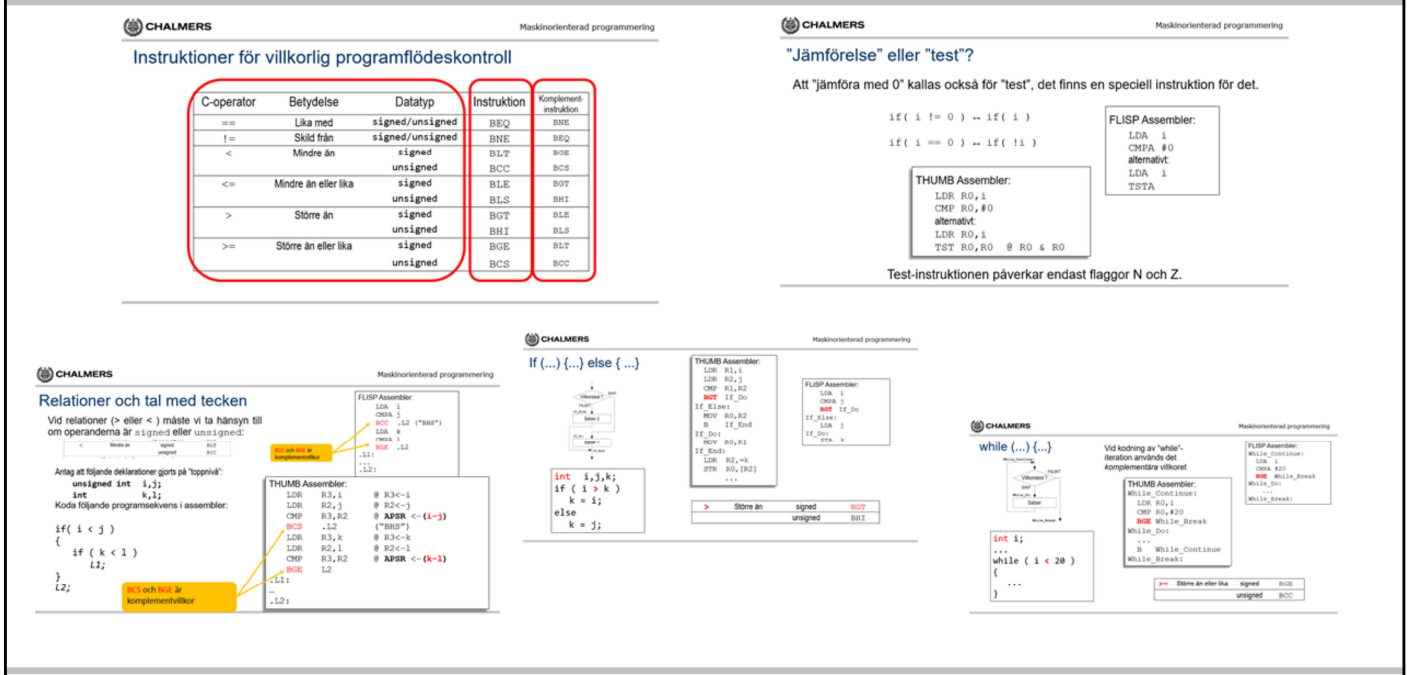
Register är generella, dvs. kan användas för alla typer av heltal på ett ganska godtyckligt sätt. Speciella så kallade “kodningskonventioner” anvisar speciella sätt att använda registeruppsättningen. Dvs. vilka register vi väljer i första hand i olika situationer. Konventionerna härstammar ofta från de tankar man hade redan då processorn en gång konstruerades.
Kompilatorkonstruktörar tillämpar oftast konventionerna. Assemblerprogrammeraren gör klokt i att följa samma konventioner.

Med ett enkelt exempel illustrerar vi ett funktionsanrop.

```
int add3( int a, int b, int c)
{
return a+b+c;
}

int main( void)
{
int result = add3( 1,2,3 );
}
```

Hur kommer det sig att variabler också har plats i minnet? Borde ju inte behövas....



Villkorliga programflödesinstruktioner (Bcc, där cc står för något villkor “condition codes”) används för att koda jämförelser och test.

Jämförelse med 0 kallas också “test”

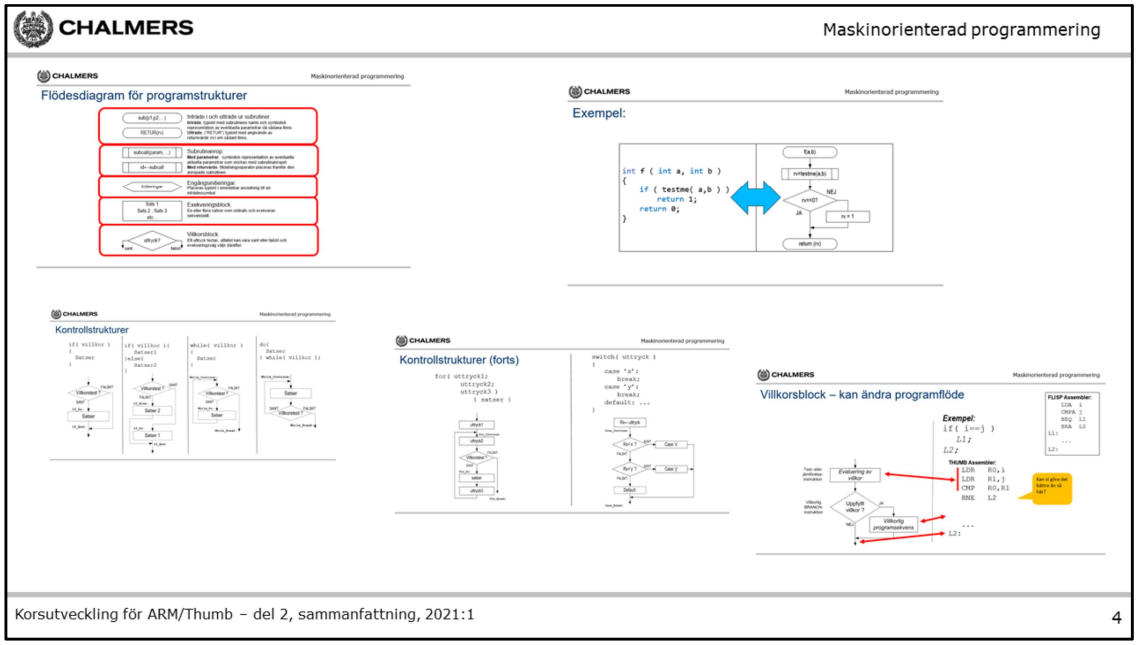
Med relationsoperatorer måste även typens tecken (signed eller unsigned) beaktas.

“Komplementära villkor” kan i vissa fall användas för att eliminera en “B” (branch-) instruktion

Grunden för kodningen ges av de båda fallen
if/else och while

Exempel, signed/unsigned, vi tittar på koden som genereras.

```
int min(int a, int b )
{
	if( a < b )
		return a;
	return b;
}
unsigned int umin( unsigned int a, unsigned int b )
{
	if( a < b )
		return a;
	return b;
}
```



Användning av flödesdiagram ger ett språkoberoende sätt att beskriva en detaljerad algoritm.
Grundläggande styrstrukturer för villkorliga programflöden är: `if/else`, while
Med dessa kan övriga syntetiseras: for-loop, do/while, if (if/else utan else-gren)

Flervalskonstruktion:
if( )
else if
...
else

Den speciella switch/case. konstruktionen används också för flera alternativa val.
Varför har man en speciell språkkonstruktion för att bekriva flerval?
Vi illustrerar med exempel:

```
int     select1( int choice )
{
        if( choice == 1 )
                return 10;
        else if (choice == 2 )
                return 20;
        else if (choice == 3 )
                return 30;
        else if (choice == 4 )
                return 40;
        else if (choice == 5 )
                return 50;
        else if (choice == 6 )
                return 60;
        else if (choice == 7 )
                return 70;
        else
                return 80;
}
respektive
int     select2( int choice )
{
        switch( choice )
        {
                case 1: return 10;
                case 2: return 20;
                case 3: return 30;
                case 4: return 40;
                case 5: return 50;
                case 6: return 60;
                case 7: return 70;
                default: return 80;
        }
}
I exemplet ger selecct1 44 st. Assemblerinstruktioner medan select2 get 28 instruktioner.
```