

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

[illegible]

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. Kompilatorkonstruktörer 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....

CHALMERS Maskinorienterad programmering

Instruktioner för villkorlig programflödeskontroll

C-operator	Betydelse	Datatyp	Instruktion	Komplement-instruktion
==	Lika med	signed/unsigned	BEQ	BNE
!=	Skilld från	signed/unsigned	BNE	BEQ
<	Mindre än	signed	BLT	BGT
<=	Mindre än eller lika	signed	BLE	BGT
>	Större än	unsigned	BLS	BHI
>=	Större än eller lika	unsigned	BHI	BLS
		signed	BGT	BLE
		unsigned	BCC	BCC

CHALMERS Maskinorienterad programmering

"Jämförelse" eller "test"?

Att "jämföra med 0" kallas också för "test", det finns en speciell instruktion för det.

```
if ( i != 0 ) -- if ( i )
if ( i == 0 ) -- if ( !i )
```

FLISP Assembler:

```
LDA i
CMPA #0
alternativt:
LDA i
TSTA
```

THUMB Assembler:

```
LDR R0, i
CMP R0, #0
alternativt:
LDR R0, i
TST R0, R0 @ R0 & R0
```

Test-instruktionen påverkar endast flaggor N och Z.

CHALMERS Maskinorienterad programmering

Relationer och tal med tecken

Vid relationer (> eller <) måste vi ta hänsyn till om operanderna är signed eller unsigned:

Antag att följande deklarationer gjorts på "toppen":

```
unsigned int i, j;
int k, l;
```

Koda följande programsekvens i assembler:

```
if ( i < j )
{
    if ( k < l )
        l++;
}
else
    l++;
```

FLISP Assembler:

```
LDA i
CMPA j
BCC #12 ("BCC")
LDA k
CMPA l
BCC #12
LDA l
ADD #1, l
BCC #12
LDA l
ADD #1, l
```

THUMB Assembler:

```
LDR R3, i
LDR R2, j
CMP R3, R2
BCC #12 ("BCC")
LDR R3, k
LDR R2, l
CMP R3, R2
BCC #12
LDR R3, l
ADD #1, R3
BCC #12
LDR R3, l
ADD #1, R3
```

CHALMERS Maskinorienterad programmering

If (...) { ... } else { ... }

THUMB Assembler:

```
LDR R1, i
LDR R2, j
CMP R1, R2
BGT If_Do
If_Do:
    MOV R0, R2
    B If_End
If_End:
    MOV R0, R1
    LDR R2, k
    STR R0, R2
    ...
```

FLISP Assembler:

```
LDA i
CMPA j
BGT If_Do
If_Do:
    LDA k
    STRA #0, k
    ...
```

> Större än signed BGT
< Större än unsigned BGT

CHALMERS Maskinorienterad programmering

while (...) { ... }

Vid kodning av "while", iteration används det komplementära villkoret

THUMB Assembler:

```
While_Continue:
    LDR R0, i
    CMP R0, #20
    BGT While_Break
    ...
    B While_Continue
While_Break:
```

FLISP Assembler:

```
While_Continue:
    LDA i
    CMPA #20
    BGT While_Break
    ...
    B While_Continue
While_Break:
```

> Större än eller lika signed BGT
< Större än eller lika unsigned BGT

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;
}
```

CHALMERS

Maskinorienterad programmering

CHALMERS

Maskinorienterad programmering

Flödesdiagram för programstruktur

init

init (start) - startpunkt för programmet

process

process - utför en eller flera instruktioner

decision

decision - utför en villkorskontroll och bestämmer vilken gren som tas

loop

loop - utför en eller flera instruktioner flera gånger

exit

exit - avslutar programmet

CHALMERS

Maskinorienterad programmering

Exempel:

```

int f ( int a, int b )
{
    if ( test(a,b) )
        return 1;
    return 0;
}

```

CHALMERS

Maskinorienterad programmering

Kontrollstrukturer

```

// while loop
while ( villkor )
{
    // instruktion
}

// do-while loop
do
{
    // instruktion
} while ( villkor );

// for loop
for ( init; villkor; inkrement )
{
    // instruktion
}

```

CHALMERS

Maskinorienterad programmering

Kontrollstrukturer (forts)

```

// switch/case
switch ( uttryck )
{
    case "a":
        // instruktion
    case "b":
        // instruktion
    default:
        // instruktion
}

```

CHALMERS

Maskinorienterad programmering

Villkorsblock - kan ändra programflöde

```

// Example: if (i == 3)
if ( i == 3 )
{
    // instruktion
}

```

CHALMERS

Maskinorienterad programmering

Flersvetskonstruktion:

```

if ( )
else if
...
else

```

CHALMERS

Maskinorienterad programmering

Den speciella switch/case. konstruktionen används också för flera alternativa val.

Varför har man en speciell språkkonstruktion för att beskriva 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 select1 44 st. Assemblerinstruktioner medan select2 ger 28 instruktioner.

```