

# Övning 3-VT2022

## Två ställen för information

När man letar info till labben finns två primära källor: Handboken och manualen.

1. Manualen finns här:

[https://ilearn.dsv.su.se/pluginfile.php/282549/mod\\_resource/content/9/DE2\\_MediaComp\\_manual.pdf](https://ilearn.dsv.su.se/pluginfile.php/282549/mod_resource/content/9/DE2_MediaComp_manual.pdf)

Hitta delen om sjusegmentsdisplayerna, sök på var man hittar minnesadresserna och den mer generella informationen.

2. Handboken finns här:

[https://ilearn.dsv.su.se/pluginfile.php/282551/mod\\_resource/content/3/n2cpu\\_nii5v1.pdf](https://ilearn.dsv.su.se/pluginfile.php/282551/mod_resource/content/3/n2cpu_nii5v1.pdf)

## Eller Instruction Set Reference

[https://ilearn.dsv.su.se/pluginfile.php/282552/mod\\_resource/content/2/n2cpu\\_nii51017.pdf](https://ilearn.dsv.su.se/pluginfile.php/282552/mod_resource/content/2/n2cpu_nii51017.pdf)

Kolla i innehållsförteckning att all information om hur processorn, instruktionerna och registren fungerar finns här.

Gå till Section I, 3. Programming models, Registers. Läs igenom hur registren är uppdelade, och vilka man kan använda fritt (r8 - r23). r0 är alltid noll, och registren med namn bör man inte peta på.

Gå till Section II, 8. Instruction Set Reference. I INios II finns tre format för instruktioner: **I**, **J** och **R**. **I** är för immediate-instruktioner, **R** för register-instruktioner och **J** för hopp-instruktioner. Det behövs olika format eftersom de hanterar olika typer av information. Kolla dessa i handboken och de olika fälten.

OP-koder (finns nedanför formaten) och man kan se nästan alla opkoder här. För **R**-instruktioner används också en OPX-kod. Genom OPX får man fler möjliga instruktioner, och R-instruktionerna använder delar av OPX-koden i sin operation.

## Scrolla ned till Assembler Pseudo-Instructions, Se vad en pseudo-instruktion är:

Använd **mov** som exempel: `mov r8, r9` översätts till `add r8, r9, r0`. Det egentligen är samma sak. **movia** och att den översätts till båda de instruktioner som står till höger.

## Handassemblering

För att förstå hur assembler faktiskt översätts till maskinkod handassemblerar vi några instruktioner. Då ser vi också hur de olika instruktionsformaten används.

Gå till **addi** (Section II, 8. Instruction Set Reference, Instruction Set Reference, **addi** add immediate). Med hjälp av den vi ser hur man handassemblerar följande rad (vi tar ett negativt tal för att visa att tvåkomplement används vid sådana fall):

**addi** r8, r9, -0x47 ( 0100 0111) -> (IMM16) 0000 0000 0100 0111

Omvandla -0x47 till tvåkomplementsform:

steg 1: Invers blir: 1111 1111 1011 1000

steg 2: addera 1 till inversen:

```
      1111 1111 1011 1000
+    0000 0000 0000 0001
-----
      1111 1111 1011 1001
```

A	B	IMM16	OP
01001	01000	1111 1111 1011 1001	000100

Här har vi först omvandlat -0x47 till tvåkomplementsform!

Gå till **and** och se hur man handassemblerar följande rad:

**and** r8, r9, r10

Vårt att påpeka är att man egentligen får fyra bitar per hexbit, så 3A ska vara 0011 1010, men bara sex av dessa bitar ska in i OP-fältet-

A	B	C	OPX	-	OP
01001	01010	01000	001110	00000	11 1010

Gå till **call** och se hur man handassemblerar följande rad:

**call** write\_serial

Använd write\_serial = 0x47BC91 som adress. När den här översättningen görs av assemblern används den faktiska minnesadressen där write\_serial har lagts. Observera att vi paddar med två bitar här för att få 26 bitar.

IMM26	OP
00 0100 0111 1011 1100 1001 0001	000000

## orhi

## bitwise logical or immediate into high halfword

Operation:	$rB \leftarrow rA \mid (\text{IMM16} : 0x0000)$
Assembler Syntax:	orhi rB, rA, IMM16
Example:	orhi r6, r7, 100
Description:	Calculates the bitwise logical OR of rA and (IMM16 : 0x0000) and stores the result in rB.
Exceptions:	None
Instruction Type:	I
Instruction Fields:	A = Register index of operand rA B = Register index of operand rB IMM16 = 16-bit signed immediate value

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A					B					IMM16																		0x34			

Gå till **movia** och visa hur man handassemblerar följande rad:

**movia** r8, 0xBA8382AF

Börja med att skriva ned de instruktioner som faktiskt utförs, och handassemblera dem. kolla var man hittar informationen om macron (Assembler Macros, ovanför Instruction Set Reference i del 8). (movia faktiskt sätter ihop två IMM16 till IMM32)

Vi får följande rader att handassemblera:

orhi r8, r0, %hiadj(BA8382AF)

addi r8, r8, %lo(BA8382AF)

Makrona ger %hiadj(BA8382AF) = BA84 (eftersom femtonde biten är ett -> BA83 + 1) och %lo(BA8382AF)=82AF.

(BA8382AF)<sub>16</sub> = (1011 1010 1000 0011 1000 0010 1010 1111)<sub>2</sub>

A	B	IMM16	OP
00000	01000	1011 1010 1000 0100	11 0100
01000	01000	1000 0010 1010 1111	00 0100

**movia** använder två register för att en adress har 32 bitar, men det finns inga instruktioner som kan ta 32 bitar på en gång.

**Table 8–4. Assembler Macros**

Macro	Description	Operation
%lo(immed32)	Extract bits [15..0] of immed32	immed32 & 0xFFFF
%hi(immed32)	Extract bits [31..16] of immed32	(immed32 >> 16) & 0xFFFF
%hiadj(immed32)	Extract bits [31..16] and adds bit 15 of immed32	((immed32 >> 16) & 0xFFFF) + ((immed32 >> 15) & 0x1)
%gprel(immed32)	Replace the immed32 address with an offset from the global pointer (1)	immed32 – _gp

**Note to Table 8–4:**

(1) Refer to the *Application Binary Interface* chapter of the *Nios II Processor Reference Handbook* for more information about global pointers.

## Ex. 2. Visar hur man hand assemblerar följande rad:

**movia r8, 0xD4EA7B3F**

(movia faktiskt sätter ihop två IMM16 till IMM32)

orhi r8, r0, %hiadj(label) = orhi, r8, r0, D4EA + 0 (15e biten)

addi r8, r8, %lo(label) = addi r8, r8, 7B3F

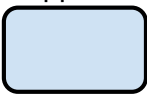

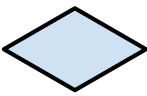
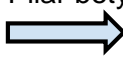

D4EA = 1101 0100 1110 1010

7B3F = 0111 1011 0011 1111

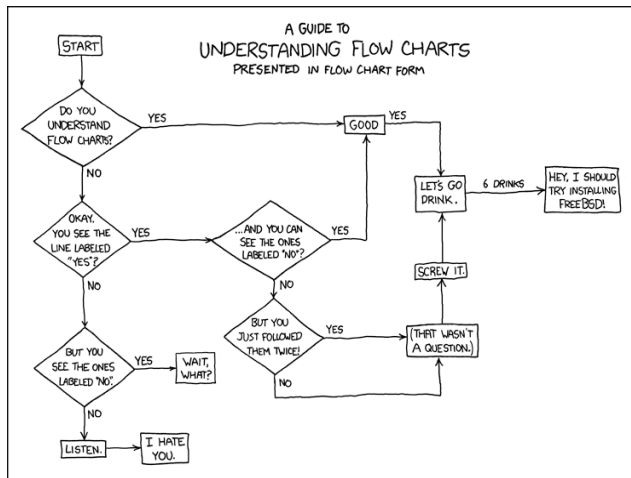
A	B	IMM16	OP
00000	01000	1101 0100 1110 1010	11 0100
01000	01000	0111 1011 0011 1111	00 0100

## Flödesdiagram

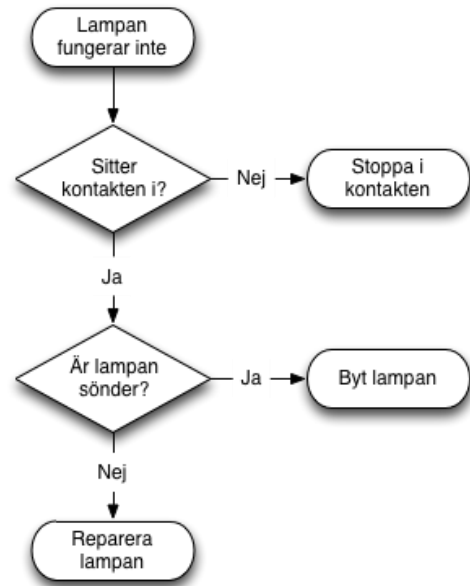
Används för att strukturera upp ett program innan man börjar koda. Väldigt användbart för att se hur ett program ska fungera. Det går dock att modellera flödet för annat också, som till exempel vad man ska göra om lampan går sönder. Rita upp och använd för att visa vad de olika formerna betyder.

<p>Rektanglar med rundade hörn: Start och stopptillstånd</p>  <p>Rektanglar med kantiga hörn: saker som ska göras (mellansteg, så att säga)</p>  <p>Romber betyder val (if-satser)</p> 	<p>Pilar betyder förflyttningar i flödet.</p>  <p>Rektanglar med dubbla kanter betyder subrutin:</p> 
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

## Exempel:



Ett annat exempel:  
<http://xkcd.com/518/>



## JNiosEmu och köra ett litet program

Vi använder samma instruktioner i JNiosEmu som vi kommer att använda på labben. JNiosEmu kan hämtas från följande länk: <http://code.google.com/p/jniosemu/>

Dock finns inte sjustementsdisplayerna, men man i teorin göra alla bitar eftersom man kan läsa av registren.

```

.text                                # Executable code follows
.global _start
_start:

    # initialize base addresses of parallel ports
    movia r15, 0x10000040            # Dip switches base address
    movia r16, 0x10000000            # red LED base address
                                      #for green led is 0x10000010

display_leds:
    ldwio r8, 0(r15)                # load slider switches
    stwio r8, 0(r16)                # write to red LEDs
    br display_leds

Kommentarerna skrivs med #

```

