Tiedonsiirtokäskyt		
LOAD	LOAD-käsky toimii jälkimmäisestä operandista ensimmäiseen. Ensimmäisen operandin pitää olla rekisteri, toinen voi olla rekisteri, vakio tai muistiosoite (myös muuttujat ovat muistiosoitteita).	
	LOAD R1, R2	Sijoitetaan arvo rekisteristä R2 menee rekisterin R1 arvoksi.
	LOAD R1, =2	Sijoitetaan rekisteriin R1 arvo 2.
	LOAD R1, 100	Sijoitetaan rekisteriin R1 se arvo, joka löytyy muistipaikasta 100.
STORE	STORE-käsky toimii ensimmäisestä operandista jälkimmäiseen Ensimmäisen operandin pitää olla rekisteri ja toisen kohdeosoite muistissa.	
	STORE R1, X	Arvo rekisteristä R1 sijoitetaan muuttujan X arvoksi.
	STORE R1, 100	Rekisterin R1 arvo sijoitetaan muistipaikkaan 100.
	EI näin: **STORE R1, =100** **STORE R1, R2**	Nämä käskyt ovat laittomia, STORE-käskyn pitää kirjoittaa muistiin. ”STORE R1, =100” tarkoittaisi suunnilleen sitä, että kirjoitettaisiin muistipaikan ”100” osoitteen päälle rekisterissä oleva arvo. Missä ei tietenkään ole mitään järkeä. ☺
IN	**IN R1, =KBD**	Lukee toisena operandina annetulta laitteelta ja tallentaa annettuun rekisteriin. TTK-91:ssä ei ole muita IN-laitteita kuin **K**ey**B**oar**D**, eli KBD.
OUT	**OUT R1, =CRT**	Tulostaa ensimmäisenä annetussa rekisterissä olevan arvon jälkimmäisenä annetulle laitteelle. TTK-91:ssä ei ole muita OUT-laitteita kuin näyttö (CRT tulee sanoista **C**athode **R**ay **T**ube).

Aritmeettiset ja loogiset käskyt

(Käytännössä loogisista käskyistä käytetään yleensä vain COMP-käskyä.)

ADD	Laskee operandien arvot yhteen ja tallentaa tuloksen ensimmäisenä annettuun rekisteriin. Toinen operandi voi olla rekisteri, vakio tai muistiosoite.	
	ADD R1, R2	R1 = R1 + R2
	ADD R1, =100	R1 = R1 + 100
	ADD R1, 100	R1 = R1 + mem[100]
	ADD R1, X	R1 = R1 + X
SUB	Vähentää ensimmäisen operandin arvosta toisen operandin arvon ja tallentaa tuloksen ensimmäisenä annettuun rekisteriin. Toinen operandi voi olla rekisteri, vakio tai muistiosoite.	
	SUB R1, R2	R1 = R1 - R2
	SUB R1, =100	R1 = R1 - 100
	SUB R1, 100	R1 = R1 - mem[100]
	SUB R1, X	R1 = R1 - X
MUL	Kertoo operandien arvot keskenään ja tallentaa tuloksen ensimmäisenä annettuun rekisteriin. Toinen operandi voi olla rekisteri, vakio tai muistiosoite. Varo ylivuotoa, jos kerrot keskenään isoja lukuja!	
	MUL R1, R2	R1 = R1 x R2
	MUL R1, =100	R1 = R1 x 100
	MUL R1, 100	R1 = R1 x mem[100]
	MUL R1, X	R1 = R1 x X
DIV	Jakaa ensimmäisen operandin arvon toisen operandin arvolla ja tallentaa tuloksen ensimmäisenä annettuun rekisteriin. Toinen operandi voi olla rekisteri, vakio tai muistiosoite. HUOM! Jakolaskussa jakojäännöstä ei talleteta mihinkään, jos tarvitset jakojäännöksen, käytä komentoa MOD.	
	DIV R1, R2	R1 = R1 / R2
	DIV R1, =100	R1 = R1 / 100
	DIV R1, 100	R1 = R1 / mem[100]
	DIV R1, X	R1 = R1 / X

MOD	Mod eli modulo. Jakaa ensimmäisen operandin arvon toisen operandin arvolla ja tallentaa jakojäännöksen ensimmäisenä annettuun rekisteriin. Toinen operandi voi olla rekisteri, vakio tai muistiosoite.	
	MOD R1, R2	R1 = R1 mod R2
	MOD R1, =100	R1 = R1 mod 100
	MOD R1, 100	R1 = R1 mod mem[100]
	MOD R1, X	R1 = R1 mod X
NOT	NOT on bittitason EI-operaatio. Se muuttaa bitin kerrallaan ykkösestä nollaan ja nollasta ykköseen	
	NOT 10101010	**= 01010101**
AND	AND on bittitason JA-operaatio. Se vertaa sanoja bitti kerrallaan, jos molemmat ovat 1, tulos on 1, muuten tulos on 0.	
	AND 11001100, 10101010	**11001100** **10101010** **----------** **10001000**
OR	OR on bittitason TAI-operaatio. Se vertaa sanoja bitti kerrallaan, jos molemmat ovat 0, tulos on 0, muuten tulos on 1.	
	OR 11001100, 10101010	**11001100** **10101010** **----------** **11101110**
XOR	XOR on bittitason poissulkeva TAI-operaatio. Poissulkevassa TAI-operaatiossa 1 ja 1 on 0, 0 ja 0 on 0 sekä 1 ja 0 on 1.	
	XOR 11001100, 10101010	**11001100** **10101010** **----------** **01100110**
SHL	SHL tulee sanoista ”shift left”. Se siirtää rekisterin Rj sisältämiä bittejä vasemmalle toisen operandin ilmoittaman määrän. Täyttää oikeaa päätä 0-biteillä.	
	SHL 11001100, =2	**= 1100110000**
SHR	SHR tulee sanoista ”shift right”. Se siirtää rekisterin Rj sisältämiä bittejä oikealle toisen operandin ilmoittaman määrän. Täyttää vasenta päätä 0-biteillä.	
	SHR 11001100, =2	**= 00110011**
COMP	Vertaa ensimmäisen operandin arvoa toisen operandin arvoon ja asettaa vertailun tuloksen tilarekisteriin (bitti L = pienempi, bitti G = suurempi, bitti E = yhtä suuri).	
	COMP R1, R2	Asettaa tilarekisterin L bitin jos R1 < R2, G bitin jos R1 > R2 ja E bitin jos R1 = R2.

Muuttujien tilanvaraus

EQU	Samaistamiskäsky, lyhenne EQU tulee sanasta ”equals”. Tunnusta voi käyttää ADDR-kentässä, jolloin kääntäjä käsittelee sen kuten vastaavaan paikkaan olisi kirjoitettu kyseinen ”arvo”.	
	Kolme EQU 3	Luodaan vakio ”Kolme”, jonka arvo on 3. Huomioi, että ”Kolme EQU =3” on väärin eikä ttk-91 ymmärrä sitä, vaikka muualla yhtäsuuruusmerkkiä käytetäänkin merkitsemään, että kyseessä on arvo (esim. ADD R1, =3 → lisää rekisterin R1 arvoon 3).
DC	DC tulee sanoista ”data constant”, mikä saattaa hämmentää joitain lukijoita. ”Constant” tarkoittaa vakiota, mutta DC-käsky ei määrittele vakiota sen yleisesti tunnetussa merkityksessä, vaan pikemminkin muuttujan. Vakioitahan ei saa muuttaa. Ttk-91 kielessä korkean tason kielen vakiota vastaa EQU-komennolla luotu tunnus. DC-käskyllä luodun tunnuksen arvoa saa muuttaa. Vertaa Javan ”int x” (DC) ja ”final int x” (EQU). Kaikki muuttujat täytyy ”luoda” tällä komennolla ennen kuin niitä voi käyttää.	
	X DC 0 **IN R1, =KBD** **STORE R1, X**	Luodaan muuttuja X, jonka alkuarvoksi alustetaan 0. Sitten luetaan näppäimistöltä arvo rekisteriin R1, ja talletetaan se muuttujaan X.
DS	DS tulee sanoista ”data segment” eli data-alue. Tämä varaa muistista käskyssä määritellyn määrän tilaa, esim. taulukkoa tai tietuetta varten. Taulukkoon viitattaessa viitataan sen ensimmäiseen alkioon. Katso luentomateriaalia taulukoiden ja tietuiden käytöstä.	
	Taulukko DS 20	Varaa muistista tilaa taulukolle, jossa on 20 alkiota. Huomioi, että tämä ei alusta taulukon alkioita vaan se pitää tehdä erikseen.

Haarautumiskäskyt		
JUMP	Ehdoton hyppy, eli sanoi tilarekisteri tai mikä vain mitä vain, siirry annettuun osoitteeseen. Annettu osoite voidaan nimetä miten vain. Voidaan käyttää esimerkiksi toteuttamaan korkean tason kielen switch-rakenteen ”break” käsky.	
	JUMP loppu	Koodissa voisi olla tällainen rivi: **loppu SVC SP, =halt** Tällöin käsky **JUMP loppu** hyppäisi suoraan tähän käskyyn, ja ohjelman suoritus loppuisi.
JNEG	Jump if negative, eli hyppää jos annetun rekisterin arvo on negatiivinen.	
	JNEG R1, negative	Jos R1 < 0, hyppää kohtaan ”negative”.
JZER	Jump if zero, eli hyppää jos annetun rekisterin arvo on nolla.	
	JZER R1, isZero	Jos R1 = 0, hyppää kohtaan ”isZero”.
JPOS	Jump if positive, eli hyppää jos annetun rekisterin arvo on positiivinen.	
	JPOS R1, positive	Jos R1 > 0, hyppää kohtaan ”positive”.
JNNEG	Jump if not negative, eli hyppää jos annetun rekisterin arvo ei ole negatiivinen. Ero JPOS-käskyyn on siinä, että ”ei negatiivinen” sisältää myös nollan.	
	JNNEG R1, notNegative	Jos R1 >= 0, hyppää kohtaan ”notNegative”.
JNZER	Jump if not zero, eli hyppää jos annetun rekisterin arvo ei ole nolla.	
	JNZERO R1, notZero	Jos R1 =/= 0, hyppää kohtaan ”notZero”.
JNPOS	Jump if not positive, eli hyppää jos annetun rekisterin arvo ei ole positiivinen. Ero JNEG-käskyyn on siinä, että ”ei positiivinen” sisältää myös nollan.	
	JNPOS R1, notPositive	Jos R1 =< 0, hyppää kohtaan ”notPositive”.
JLES	Jump if less, eli hyppää jos pienempi. Tämä ja viisi seuraavaa käskyä tutkivat tilarekisterin L, G ja E bittejä. Niitä käytetään COMP-käskyn yhteydessä. Kannattaa huomioida, että myös muut operaatiot saattavat muuttaa tilarekisteriä, joten vertailu COMP-käskyllä kannattaa suorittaa juuri ennen ehdollista hyppykäskyä.	

	LOAD R1, X
	COMP R1, =2
	JLES else
	Tee jotain jos X >=2
		if (x >= 2) {
		  tee jotain
		}
		else {

	else Tee jotain muuta jos X < 2	tee jotain muuta }
JEQU	Jump if equal, eli hyppää jos yhtä suuri.	
	Loop IN R1, =KBD **COMP R1, =0** **JEQU Loppu** **OUT R1, =CRT** **JUMP Loop** **Loppu SVC SP, =halt**	Yksinkertainen pikku ohjelma, joka lukee näppäimistöltä syötteitä rekisteriin R1. Jos se löytää nollan, se hyppää ”Loppu” kohtaan ja lopettaa. Muuten hyppää lukemaan uuden syötteen.
JGRE	Jump if greater, eli hyppää jos suurempi.	
	COMP R1, =3 **JGRE** **suurempiKuinKolme**	Hyppää jos R1:n arvo on suurempi kuin kolme.
JNLES	Jump if not less, eli hyppää jos ”ei pienempi”, siis yhtä suuri tai suurempi.	
	COMP R1, =5 **JNLES viisiTaiSuurempi**	Hyppää jos R1:n arvo on suurempi tai yhtä suuri kuin viisi.
JNEQU	Jump if not equal, eli hyppää jos eri suuri.	
	Loop IN R2, =KBD Tee jotain **COMP R2, =-1** **JNEQU Loop**	Voitaisiin käyttää vaikkapa silmukassa, jossa pyöritään kunnes saadaan syöte ”-1”.
JNGRE	Jump if not greater, eli hyppää jos ”ei suurempi”, siis yhtä suuri tai pienempi.	
	COMP R2, =5 **JNGRE viisiTaiPienempi**	Hyppää jos R2:n arvo on pienempi tai yhtä suuri kuin viisi.

Pinokäskyt		
PUSH	”Pushaa” eli laittaa pinon päällimmäiseksi toisena operandina annetun alkion. Ensimmäinen operandi on aina pinorekisteri SP. Pino-osoittimen arvo kasvaa yhdellä.	
	PUSH SP, R1	Laitetaan rekisterin R1 arvo pinoon.
POP	”Poppaa” eli ottaa pinosta päällimmäisen alkion. Ensimmäinen operandi on aina pinorekisteri SP, toinen operandi on rekisteri johon pinosta otettu arvo halutaan tallentaa. Pino-osoittimen arvo vähenee yhdellä.	
	POP SP, R2	Otetaan pinon päällimmäinen alkio ja sijoitetaan se rekisteriin R2.

PUSHR	”Push registers” eli laittaa rekisterit R0 – R6 pinoon. Pino-osoittimen arvo kasvaa seitsemällä. Käytetään lähinnä aliohjelmien toteuttamisessa.	
	PUSHR SP	Tässä ei tarvita toista operandia.
POPR	”Pop registers” eli otetaan rekisterit pois pinosta. Pino-osoittimen arvo vähenee seitsemällä. Käytetään lähinnä aliohjelmien toteuttamisessa.	
	POPR SP	Tässäkään ei tarvita toista operandia.

Aliohjelmakäskyt		
CALL	Kutsuu aliohjelmaa, jonka osoite annetaan toisena operandina. Ensimmäisenä operandina on aina pino.	
	CALL SP, ali	Kutsuu aliohjelmaa nimeltä ”ali”.
EXIT	Poistuu aliohjelmasta. Ensimmäisenä operandina taas pino. Toisena operandina annetaan kutsua ennen pinoon vietyjen parametrien määrä, jotta pino-osoitin saadaan takaisin oikeaan kohtaan.	
	EXIT SP, =2	Poistutaan aliohjelma, jolla oli kaksi parametria, esim Summa(x, y).

Muut käskyt		
SVC	**S**uper**V**isor **C**all, eli käyttöjärjestelmäkutsu. Ilmoittaa käyttöjärjestelmälle, että nyt tarvittaisiin sen palveluita. Teoriassa voitaisiin käyttää esim. levylle kirjoittamiseen tai lukemiseen, mutta tämän kurssin puitteissa ainut tarvittava käsky on ohjelman pysäytys.	
	SVC SP, =halt	Pysäyttää ohjelman. Käyttöjärjestelmäkutsut ”kohdistuvat” aina pinoon kuten aliohjelmakutsut.
NOP	**”N**o **OP**eration”-käsky. Ei tee mitään, mutta varaa tilaa muistista. Käytännössä oikeastaan käytetään siihen, että saadaan data-alue alkamaan oikeasta kohdasta, että TitoTrainer voi automaattisesti tarkistaa, että tehtävä on oikein vertaamalla tulostettuja arvoja.	
	NOP	Ei tarvitse mitään operandeja.