Esperanta klubvespero

“La bazo de programado”

prezentisto: Luko

ĵaudo la 2024-05-09

https://lukas-prokop.at/talks/klubvesperoj/

Kiu mi estas?

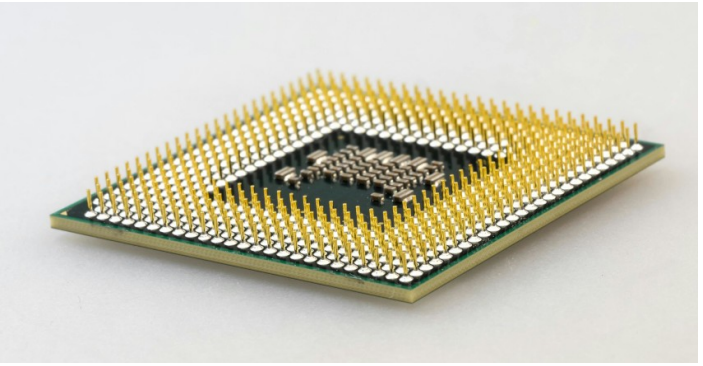
Esperanta nomo: Luko

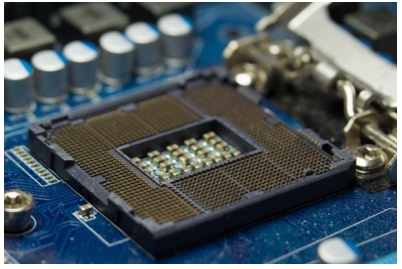
1) ĉefe mi estas programisto
2) en mia libertempo
   - mi praktikas kaj instruas Aikidō
   - mi lernas Esperanton kaj la japanan
   - mi stud(i/a)s informatikon kaj matematikon
   - mi verkas programon por cifereca kompostado

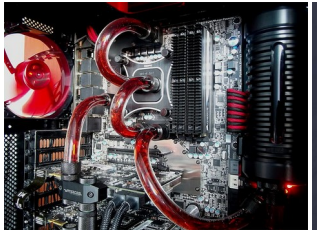
Tagordo

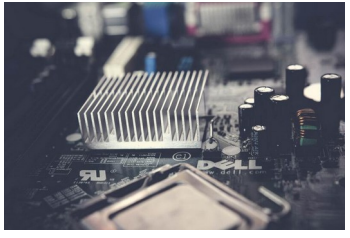
- Vortprovizo
- Elektra bazo de komputilo
- Bitoj kaj bitokoj
- Uzado de memoro
- Periferiaj aparatoj
- Ĉefprocesoro kaj traktadaj instrukcioj

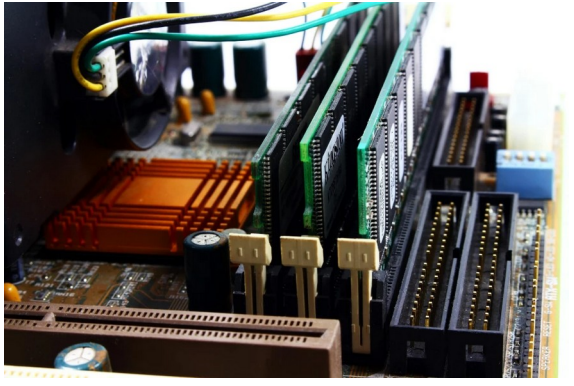
Vortprovizo





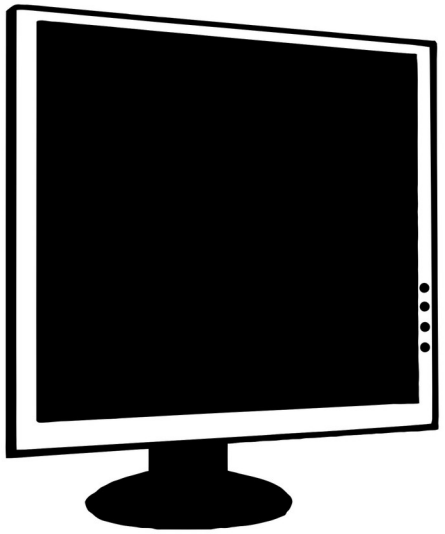


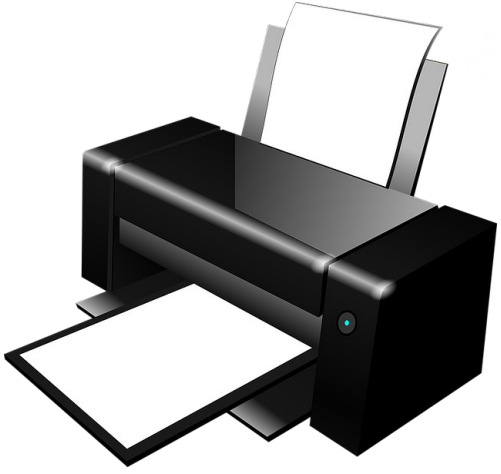




Vortprovizo

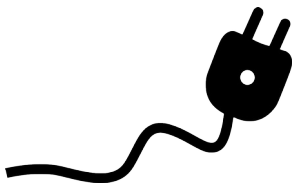
Vortprovizo



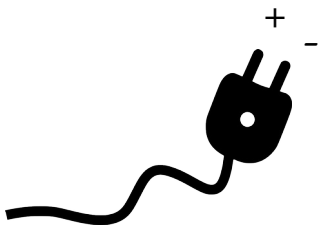


Elektra bazo de komputilo

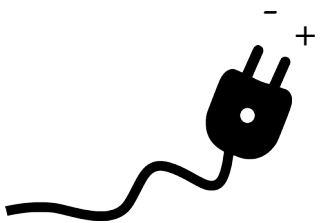
Elektra bazo de komputilo



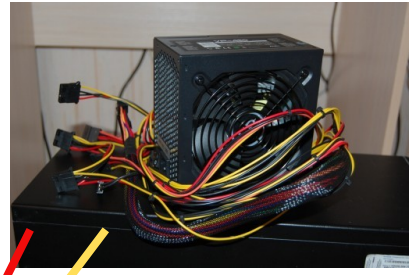
Elektra bazo de komputilo

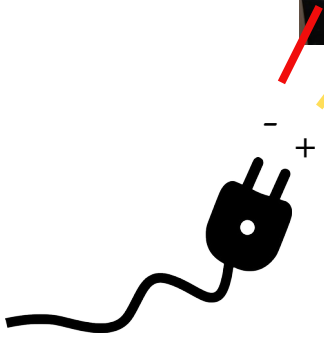


Elektra bazo de komputilo

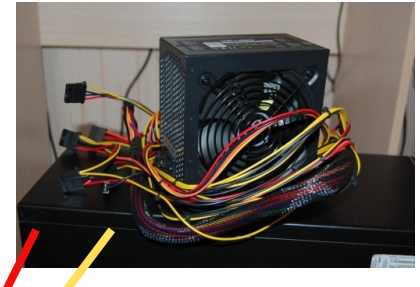


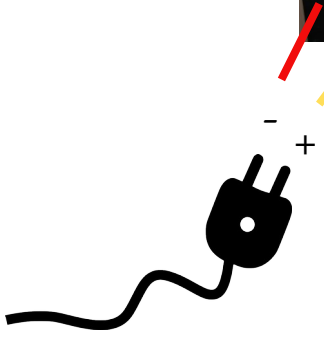
Elektra bazo de komputilo



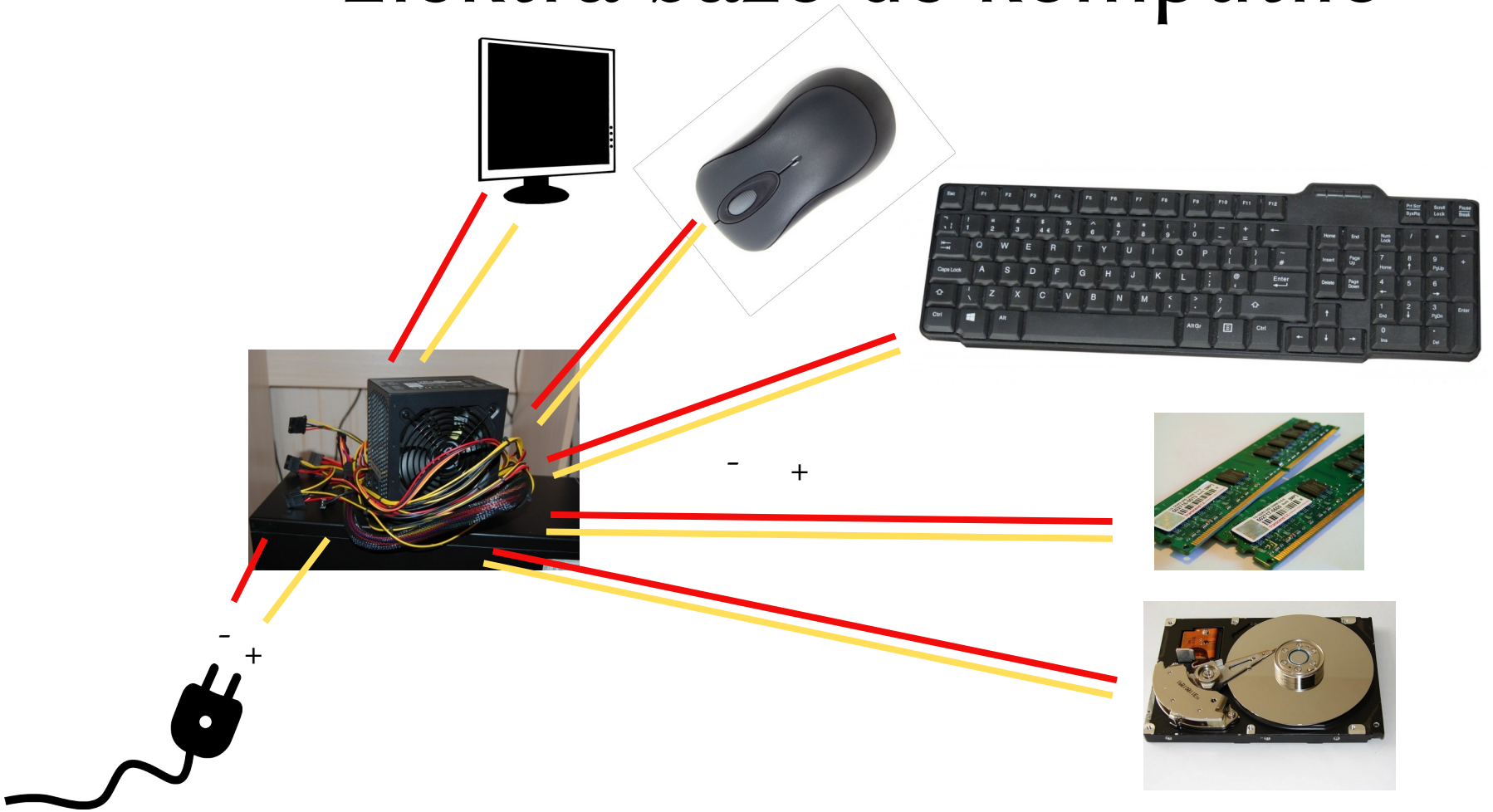


Elektra bazo de komputilo

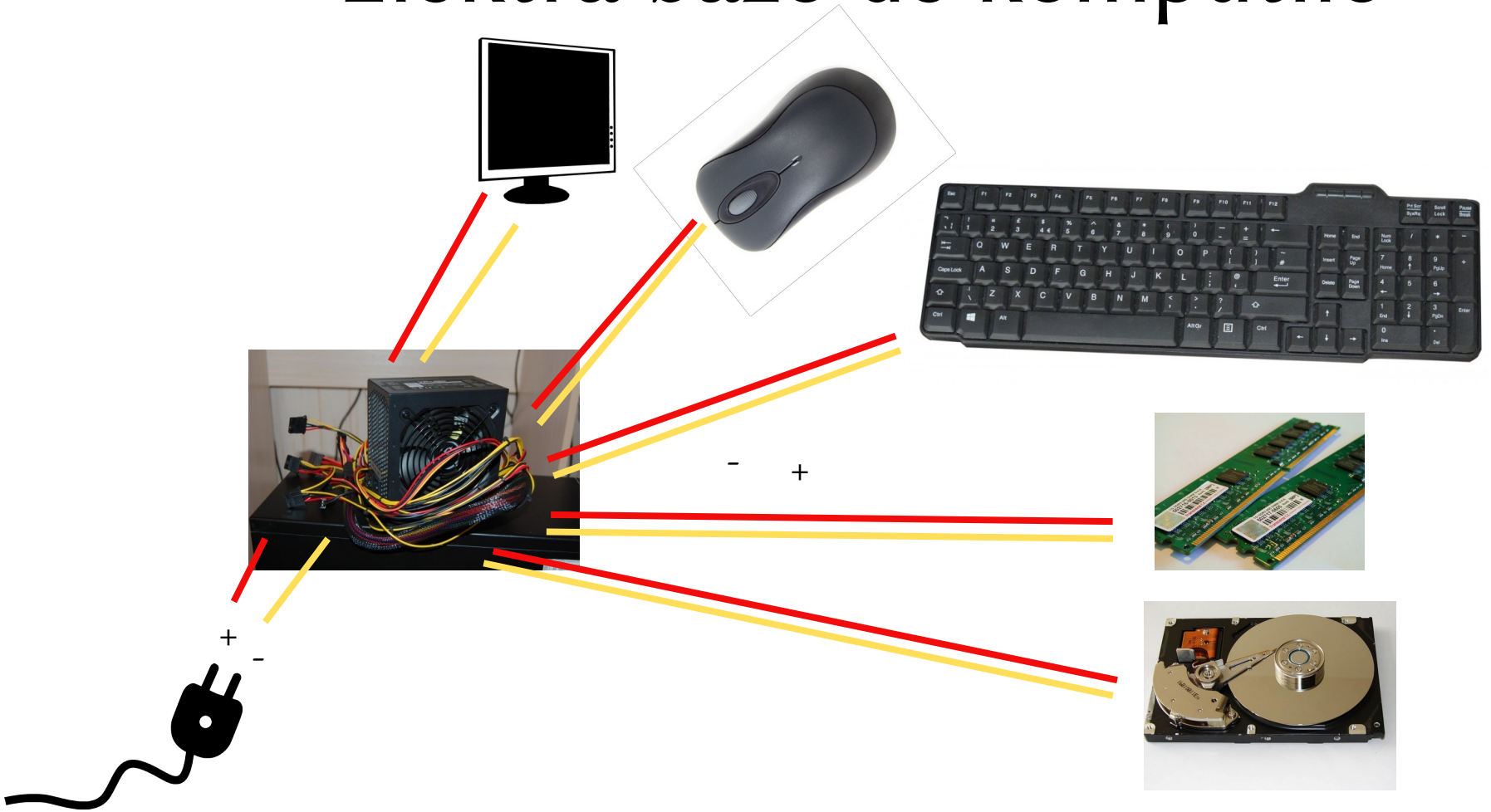




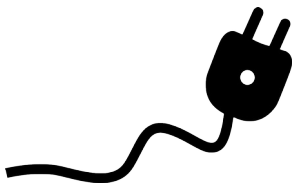
Elektra bazo de komputilo



Elektra bazo de komputilo



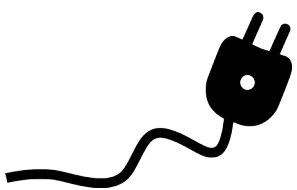
Elektra bazo de komputilo



Elektra bazo de komputilo

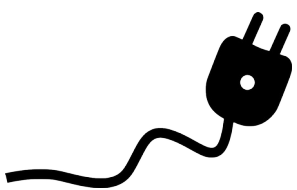
mezurilo

t	1	2	3	4	5	6	7	8	9	10	11	12	13	14



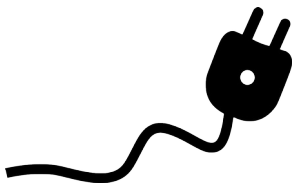
Elektra bazo de komputilo

t	1	2	3	4	5	6	7	8	9	10	11	12	13	14
	0													

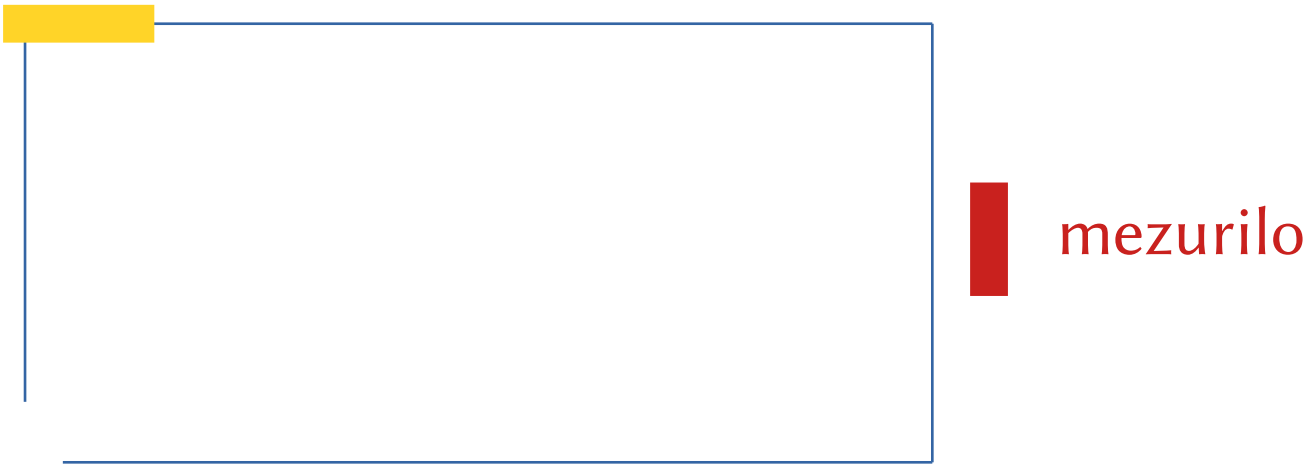


Elektra bazo de komputilo

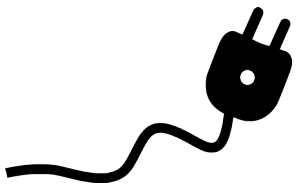
t	1	2	3	4	5	6	7	8	9	10	11	12	13	14
	0	0												



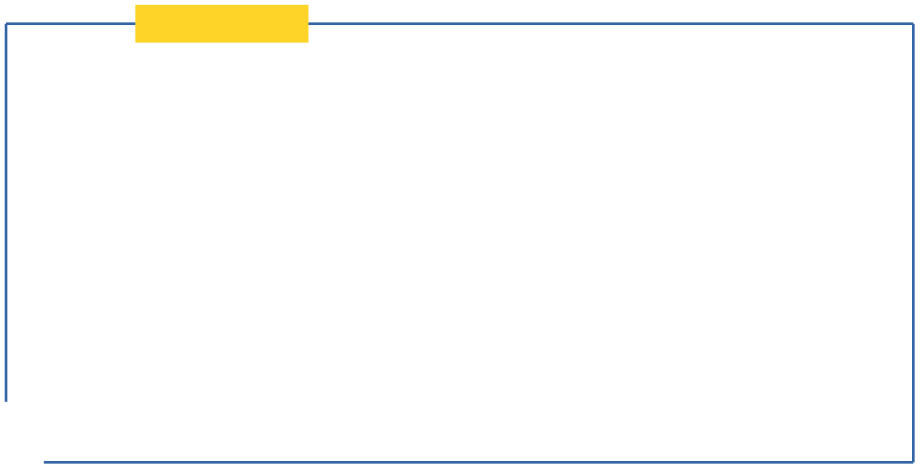
Elektra bazo de komputilo



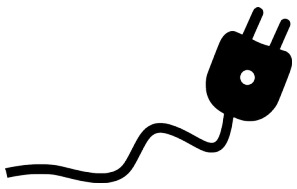
t	1	2	3	4	5	6	7	8	9	10	11	12	13	14
	0	0	0											



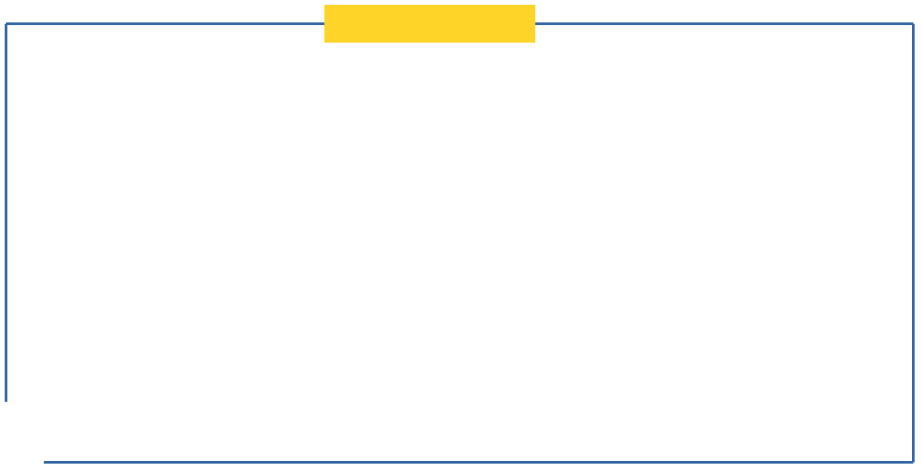
Elektra bazo de komputilo



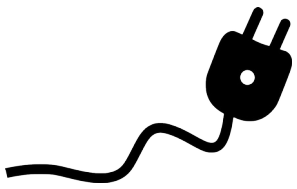
t	1	2	3	4	5	6	7	8	9	10	11	12	13	14
	0	0	0	0										



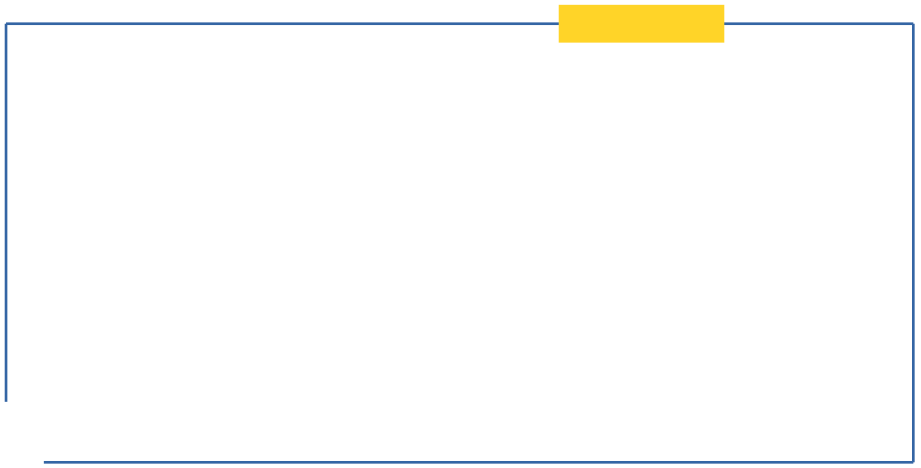
Elektra bazo de komputilo



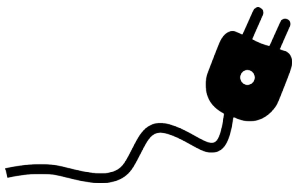
t	1	2	3	4	5	6	7	8	9	10	11	12	13	14
	0	0	0	0	0									



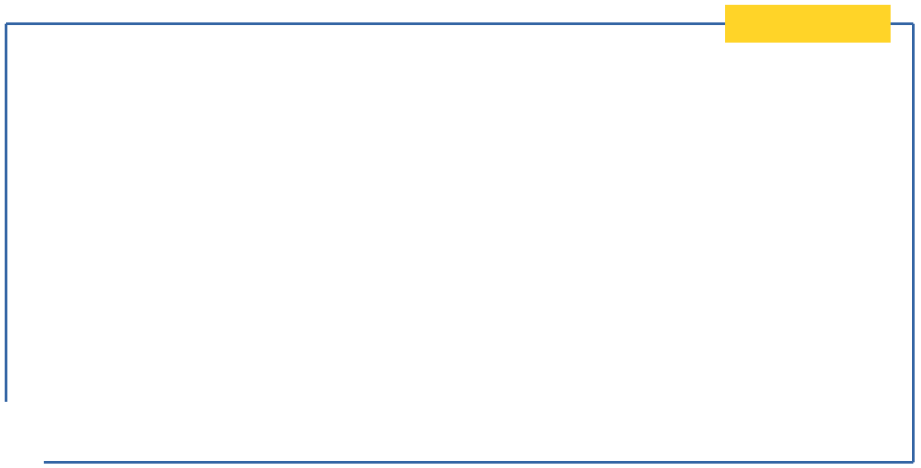
Elektra bazo de komputilo



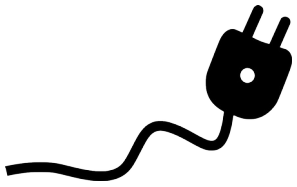
t	1	2	3	4	5	6	7	8	9	10	11	12	13	14
	0	0	0	0	0	0								



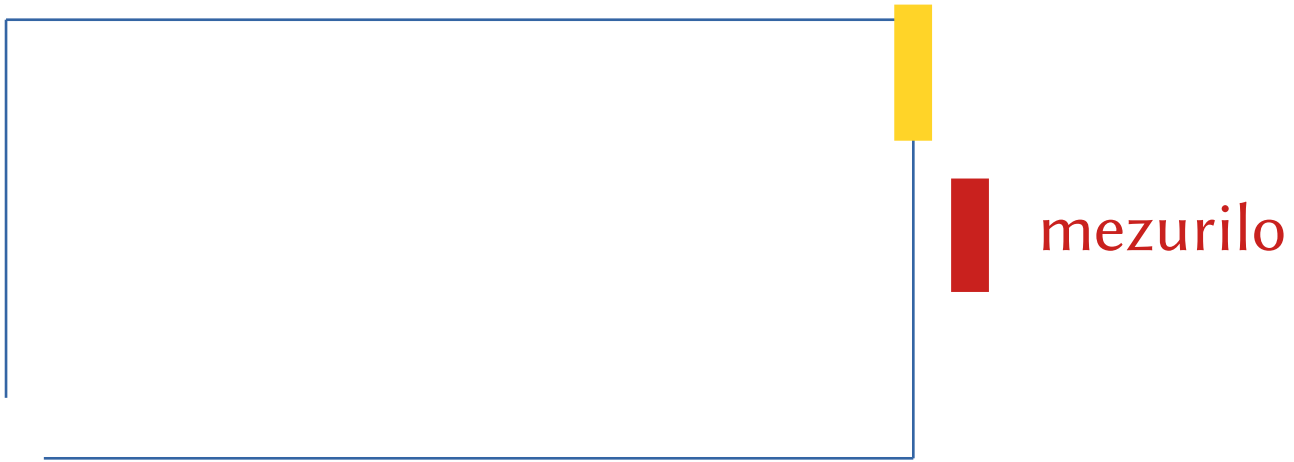
Elektra bazo de komputilo



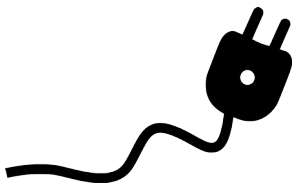
t	1	2	3	4	5	6	7	8	9	10	11	12	13	14
	0	0	0	0	0	0	0							



Elektra bazo de komputilo

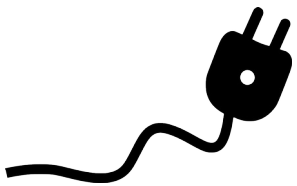


t	1	2	3	4	5	6	7	8	9	10	11	12	13	14
	0	0	0	0	0	0	0	0						



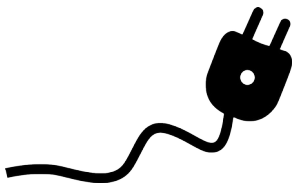
Elektra bazo de komputilo

t	1	2	3	4	5	6	7	8	9	10	11	12	13	14
	0	0	0	0	0	0	0	0	1					



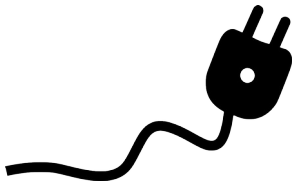
Elektra bazo de komputilo

t	1	2	3	4	5	6	7	8	9	10	11	12	13	14
	0	0	0	0	0	0	0	0	1	0				

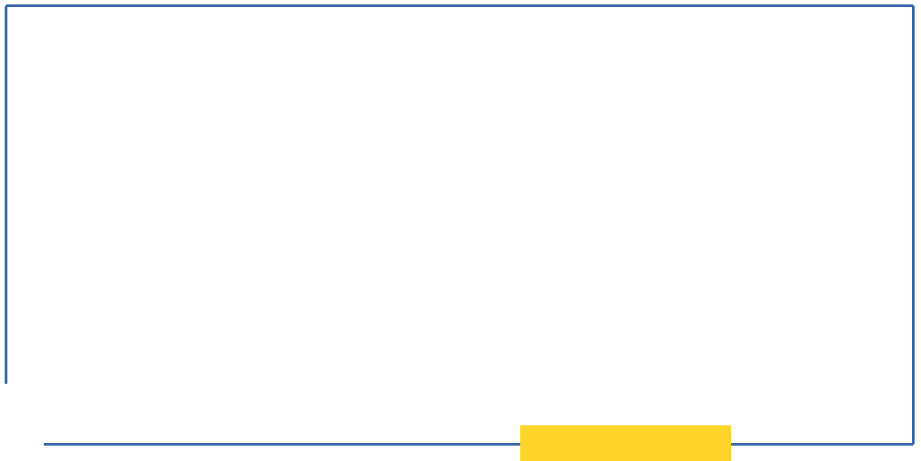


Elektra bazo de komputilo

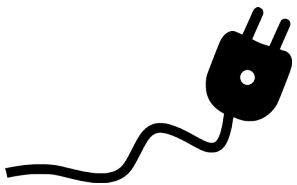
t	1	2	3	4	5	6	7	8	9	10	11	12	13	14
	0	0	0	0	0	0	0	0	1	0		0		



Elektra bazo de komputilo

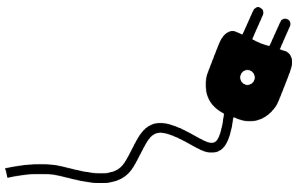


t	1	2	3	4	5	6	7	8	9	10	11	12	13	14
	0	0	0	0	0	0	0	0	1	0	0	0		



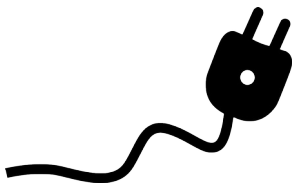
Elektra bazo de komputilo

t	1	2	3	4	5	6	7	8	9	10	11	12	13	14
	0	0	0	0	0	0	0	0	1	0	0	0	0	



Elektra bazo de komputilo

t	1	2	3	4	5	6	7	8	9	10	11	12	13	14
	0	0	0	0	0	0	0	0	1	0	0	0	0	0



Bitoj kaj bitokoj

Bitoj kaj bitokoj

bitoj / duuma (2)

0 0 0 0 0 0 0 0

dekuma (10)

0

deksesuma (16)

0

Bitoj kaj bitokoj

bitoj / duuma (2)

0 0 0 0 0 0 0 1

dekuma (10)

1

deksesuma (16)

1

Bitoj kaj bitokoj

bitoj / duuma (2)

0 0 0 0 0 0 1 0

dekuma (10)

2

deksesuma (16)

2

Bitoj kaj bitokoj

bitoj / duuma (2)

0 0 0 0 0 0 1 1

dekuma (10)

3

deksesuma (16)

3

Bitoj kaj bitokoj

bitoj / duuma (2)

0 0 0 0 0 1 0 0

dekuma (10)

4

deksesuma (16)

4

Bitoj kaj bitokoj

bitoj / duuma (2)

0 0 0 0 0 1 0 1

dekuma (10)

5

deksesuma (16)

5

Bitoj kaj bitokoj

bitoj / duuma (2)

0 0 0 0 0 1 1 0

dekuma (10)

6

deksesuma (16)

6

Bitoj kaj bitokoj

bitoj / duuma (2)

0 0 0 0 0 1 1 1

dekuma (10)

7

deksesuma (16)

7

Bitoj kaj bitokoj

bitoj / duuma (2)

0 0 0 0 1 0 0 0

dekuma (10)

8

deksesuma (16)

8

Bitoj kaj bitokoj

bitoj / duuma (2)

0 0 0 0 1 0 0 1

dekuma (10)

9

deksesuma (16)

9

Bitoj kaj bitokoj

bitoj / duuma (2)

0 0 0 0 1 0 1 0

dekuma (10)

1 0

deksesuma (16)

A

Bitoj kaj bitokoj

bitoj / duuma (2)

0 0 0 0 1 0 1 1

dekuma (10)

1 1

deksesuma (16)

B

Bitoj kaj bitokoj

bitoj / duuma (2)

0 0 0 0 1 1 0 0

dekuma (10)

1 2

deksesuma (16)

C

Bitoj kaj bitokoj

bitoj / duuma (2)

0 0 0 0 1 1 0 1

dekuma (10)

1 3

deksesuma (16)

D

Bitoj kaj bitokoj

bitoj / duuma (2)

0 0 0 0 1 1 1 0

dekuma (10)

1 4

deksesuma (16)

E

Bitoj kaj bitokoj

bitoj / duuma (2)

0 0 0 0 1 1 1 1

dekuma (10)

15

deksesuma (16)

F

Bitoj kaj bitokoj

bitoj / duuma (2)

0 0 0 1 0 0 0 0

dekuma (10)

1 6

deksesuma (16)

1 0

Bitoj kaj bitokoj

bitoj / duuma (2)

...

dekuma (10)

...

deksesuma (16)

...

Bitoj kaj bitokoj

bitoj / duuma (2)

1 1 1 1 1 1 1 1

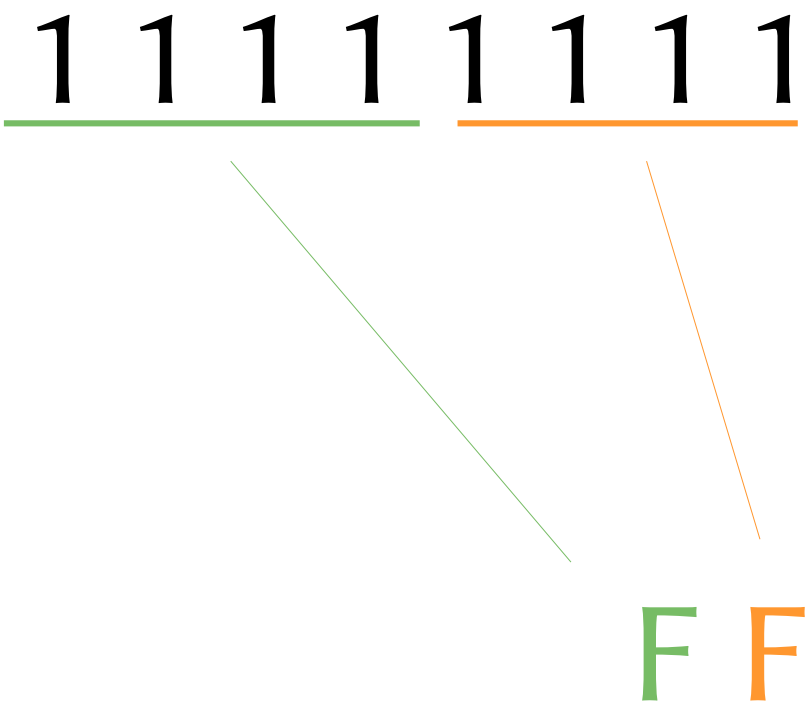
dekuma (10)

2 5 5

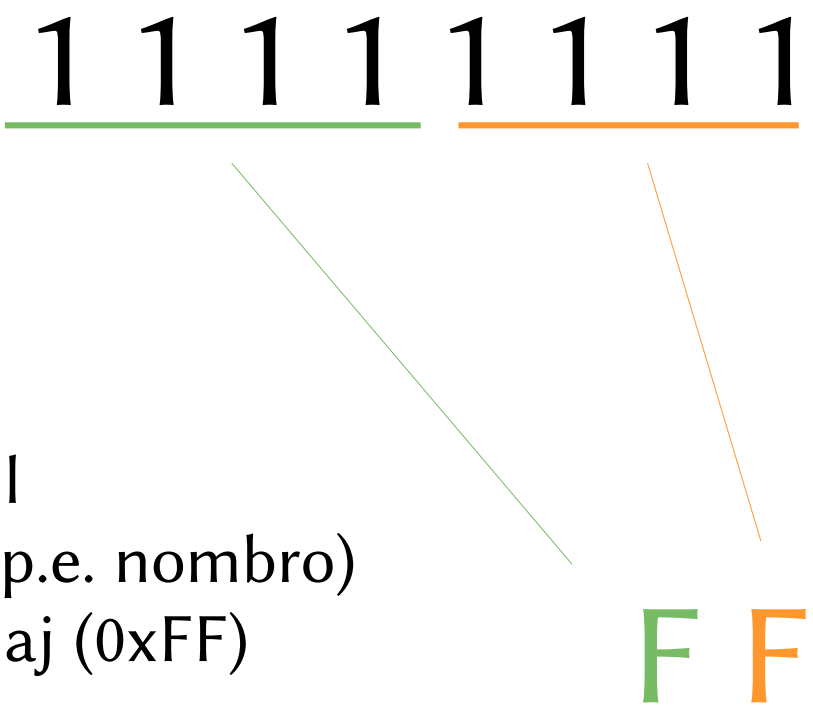
deksesuma (16)

F F

Bitoj kaj bitokoj



Bitoj kaj bitokoj



- Kutimo de programistoj:
  Ni komencas per la nombro nul
- Stato de bitoj reprezentas ion (p.e. nombro)
- Deksesumaj nombroj estas utilaj (0xFF)

Bitoj kaj bitokoj

bitoj / duuma (2)

0 0 0 0 0 0 0 0

signifo

malprava

deksesuma (16)

0

Bitoj kaj bitokoj

bitoj / duuma (2)

0 0 0 0 0 0 0 1

signifo

prava

deksesuma (16)

1

Bitoj kaj bitokoj

bitoj / duuma (2)

0 1 0 0 0 0 0 1

signifo

A

deksesuma (16)

4 1

Bitoj kaj bitokoj

bitoj / duuma (2)

0 1 0 0 0 0 1 0

signifo

B

deksesuma (16)

4 2

Bitoj kaj bitokoj

bitoj / duuma (2)

0 1 1 0 0 0 1 0

signifo

b

deksesuma (16)

6 2

Bitoj kaj bitokoj

	bulea	**entjera**	**signa**
0 0 0 0 0 0 0 0	malpr.	0	nulo
0 0 0 0 0 0 0 1	prava	1	ekrubriko
0 0 0 0 0 0 1 0	-	2	ektitolo
0 0 0 0 0 0 1 1	-	3	fintitolo
...		...	...

Bitoj kaj bitokoj

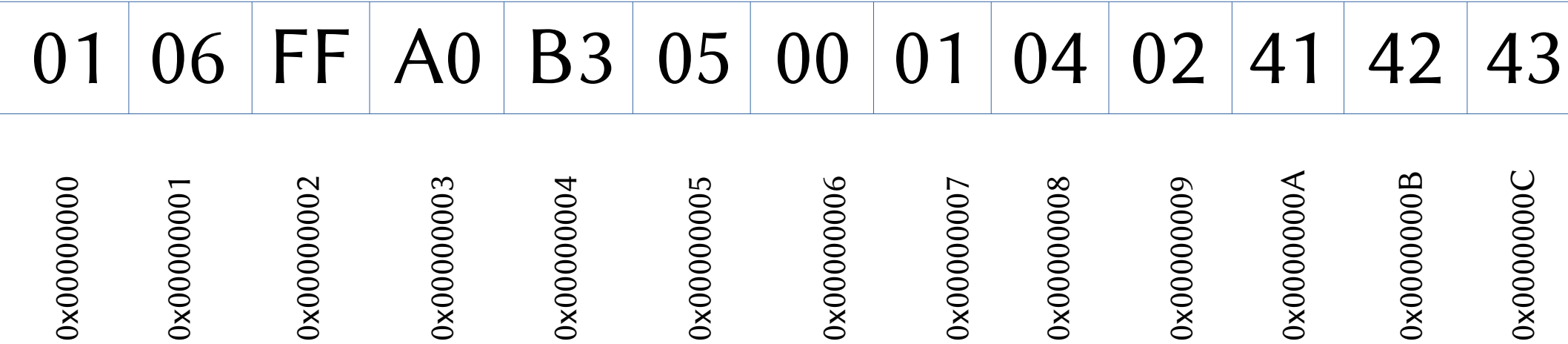
datumtipoj →	**bulea**	**entjera**	**signa**
0 0 0 0 0 0 0 0	malpr.	0	nulo
0 0 0 0 0 0 0 1	prava	1	ekkrubriko
0 0 0 0 0 0 1 0	-	2	ektitolo
0 0 0 0 0 0 1 1	-	3	fintitolo
...		...	...

Ĉia valoro estas stato de bitoj!

Uzado de memoro

01	06	FF	A0	B3	05	00	01	04	02	41	42	43

Uzado de memoro



Ni referencas valorojn (bitokojn) per *memora adreso*.

Uzado de memoro

01	06	FF	A0	B3	05	00	01	04	02	41	42	43
0x00000000	0x00000001	0x00000002	0x00000003	0x00000004	0x00000005	0x00000006	0x00000007	0x00000008	0x00000009	0x0000000A	0x0000000B	0x0000000C

Ni referencas valorojn (bitokojn) per *memora adreso*.
Instrukcio: “skribu A5 ĉe adreso 0x00000004”

Uzado de memoro

01	06	FF	A0	A5	05	00	01	04	02	41	42	43
0x00000000	0x00000001	0x00000002	0x00000003	0x00000004	0x00000005	0x00000006	0x00000007	0x00000008	0x00000009	0x0000000A	0x0000000B	0x0000000C

Ni referencas valorojn (bitokojn) per *memora adreso*.
Instrukcio: “skribu A5 ĉe adreso 0x00000004” ✓

Uzado de memoro

01	06	FF	A0	A5	05	00	01	04	02	41	42	43
0x00000000	0x00000001	0x00000002	0x00000003	0x00000004	0x00000005	0x00000006	0x00000007	0x00000008	0x00000009	0x0000000A	0x0000000B	0x0000000C

Ni referencas valorojn (bitokojn) per *memora adreso*.
Instrukcio: “skribu A5 ĉe adreso 0x00000004” ✓
Instrukcio: “adiciu 02 kaj la valoro de adreso 0x00000001 al 0x00000001”

Uzado de memoro

01	08	FF	A0	A5	05	00	01	04	02	41	42	43
0x00000000	0x00000001	0x00000002	0x00000003	0x00000004	0x00000005	0x00000006	0x00000007	0x00000008	0x00000009	0x0000000A	0x0000000B	0x0000000C

Ni referencas valorojn (bitokojn) per *memora adreso*.
Instrukcio: “skribu A5 ĉe adreso 0x00000004” ✓
Instrukcio: “adiciu 02 kaj la valoro de adreso 0x00000001 al 0x00000001” ✓

Uzado de memoro

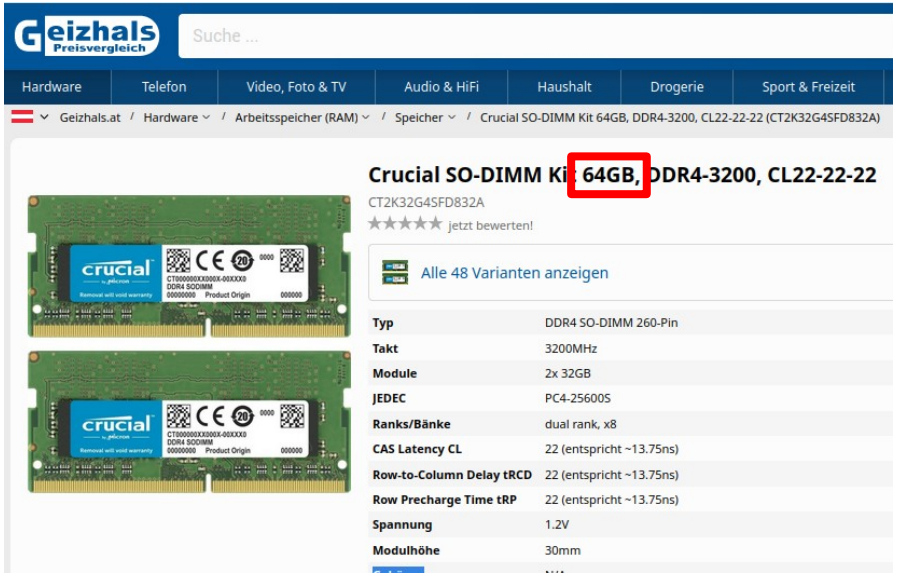
01	06	FF	A0	B3	05	00	01	04	02	41	42	43

ĉi tie:

13 bitokoj

ĉefmemoro (ekzemplo):

$64 \cdot 1024^3 = 68\,719\,476\,736$ bitokoj



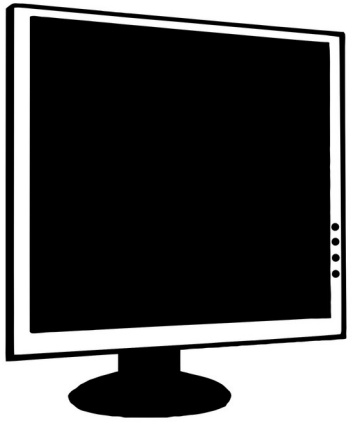
Periferiaj aparatoj

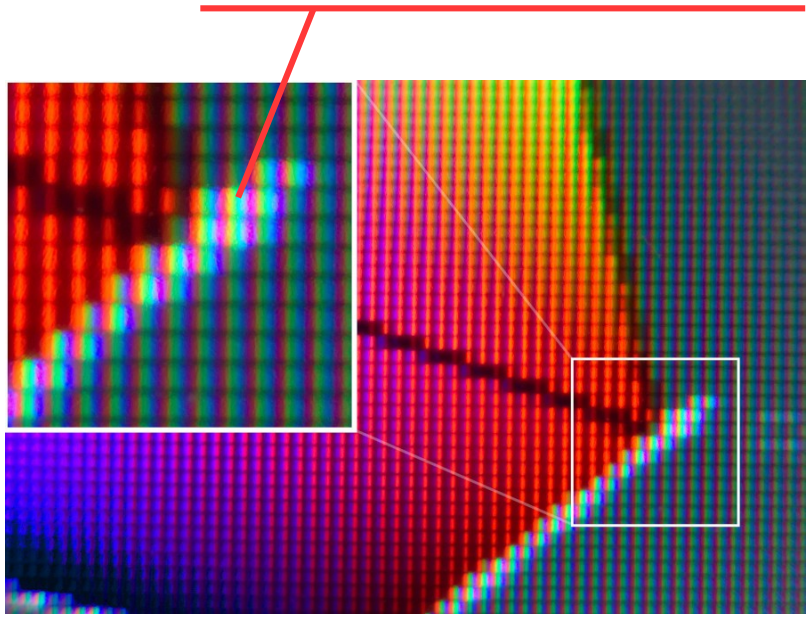
Periferiaj aparatoj

01	06	FF	A0	B3	05	00	01	04	02	41	42	43

Periferiaj aparatoj

01	06	FF	A0	B3	05	00	01	04	02	41	42	43





Ĉefprocesoro kaj traktadaj instrukcioj

Ĉefprocesoro kaj traktadaj instrukcioj

- Signaloj en bitoj
- Bito(ko)j estas en la memoro.
  Ni referencas valoroj kiel bitokoj.
- Programo nur estas vico de bitokoj en memoro
- Ni povas redakti la memoron per instrukcioj

Ĉefprocesoro kaj traktadaj instrukcioj

- Kiuj instrukcioj estas uzeblaj?

Ĉefprocesoro kaj traktadaj instrukcioj

- Kiuj instrukcioj estas uzeblaj?
  - dependas de via arĥitekturo (instrukciaro)
  - p.e. x86, x86_64, amd64, ARMv8, RISC-V, ...
  - eblecoj similas: metu valoro al adreso, kopiu de ... al ..., kajo, aŭo, komparu ... al ..., adiciu ..., multipliku ..., subtrahu ...

Ĉefprocesoro kaj traktadaj instrukcioj

Free & Open RISC-V Reference Card ①

Base Integer Instructions: RV32I, RV64I, and RV128I

Category	Name	Fmt	RV32I Base		+RV{64,128}	
Loads	Load Byte	I	LB	rd,rs1,imm		
	Load Halfword	I	LH	rd,rs1,imm		
	Load Word	I	LW	rd,rs1,imm	L{D|Q}	rd,rs1,imm
	Load Byte Unsigned	I	LBU	rd,rs1,imm		
	Load Half Unsigned	I	LHU	rd,rs1,imm	L{W|D}U	rd,rs1,imm
Stores	Store Byte	S	SB	rs1,rs2,imm		
	Store Halfword	S	SH	rs1,rs2,imm		
	Store Word	S	SW	rs1,rs2,imm	S{D|Q}	rs1,rs2,imm
Shifts	Shift Left	R	SLL	rd,rs1,rs2	SLL{W|D}	rd,rs1,rs2
	Shift Left Immediate	I	SLLI	rd,rs1,shamt	SLLI{W|D}	rd,rs1,shamt
	Shift Right	R	SRL	rd,rs1,rs2	SRL{W|D}	rd,rs1,rs2
	Shift Right Immediate	I	SRLI	rd,rs1,shamt	SRLI{W|D}	rd,rs1,shamt
	Shift Right Arithmetic	R	SRA	rd,rs1,rs2	SRA{W|D}	rd,rs1,rs2
	Shift Right Arith Imm	I	SRAI	rd,rs1,shamt	SRAI{W|D}	rd,rs1,shamt
Arithmetic	ADD	R	ADD	rd,rs1,rs2	ADD{W|D}	rd,rs1,rs2
	ADD Immediate	I	ADDI	rd,rs1,imm	ADDI{W|D}	rd,rs1,imm
	SUBtract	R	SUB	rd,rs1,rs2	SUB{W|D}	rd,rs1,rs2
	Load Upper Imm	U	LUI	rd,imm		
	Add Upper Imm to PC	U	AUIPC	rd,imm		
Logical	XOR	R	XOR	rd,rs1,rs2		
	XOR Immediate	I	XORI	rd,rs1,imm		
	OR	R	OR	rd,rs1,rs2		
	OR Immediate	I	ORI	rd,rs1,imm		
	AND	R	AND	rd,rs1,rs2		
	AND Immediate	I	ANDI	rd,rs1,imm		

RV Privileged Instructions

Category	Name	RV mnemonic
CSR Access	Atomic R/W	CSRRW rd,csr,rs1
	Atomic Read & Set Bit	CSRRS rd,csr,rs1
	Atomic Read & Clear Bit	CSRRC rd,csr,rs1
	Atomic R/W Imm	CSRRWI rd,csr,imm
	Atomic Read & Set Bit Imm	CSRRSI rd,csr,imm
	Atomic Read & Clear Bit Imm	CSRRCI rd,csr,imm
Change Level	Env. Call	ECALL
	Environment Breakpoint	EBREAK
	Environment Return	ERET
Trap Redirect	to Supervisor	MRTS
	Redirect Trap to Hypervisor	MRTH
	Hypervisor Trap to Supervisor	HRTS
Interrupt	Wait for Interrupt	WFI
MMU	Supervisor FENCE	SFENCE.VM rs1

Optional Compressed (16-bit) Instruction Extension: RVC

Category	Name	Fmt	RVC		RVI equivalent
Loads	Load Word	CL	C.LW	rd',rs1',imm	LW rd',rs1',imm*4
	Load Word SP	CI	C.LWSP	rd,imm	LW rd,sp,imm*4
	Load Double	CL	C.LD	rd',rs1',imm	LD rd',rs1',imm*8
	Load Double SP	CI	C.LDSP	rd,imm	LD rd,sp,imm*8
	Load Quad	CL	C.LQ	rd',rs1',imm	LQ rd',rs1',imm*16
	Load Quad SP	CI	C.LQSP	rd,imm	LQ rd,sp,imm*16

https://www.cl.cam.ac.uk/teaching/1617/ECAD+Arch/files/docs/RISCVGreenCardv8-20151013.pdf

Ĉefprocesoro kaj traktadaj instrukcioj

Compare	Set <	R	`SLT rd,rs1,rs2`
	Set < Immediate	I	`SLTI rd,rs1,imm`
	Set < Unsigned	R	`SLTU rd,rs1,rs2`
	Set < Imm Unsigned	I	`SLTIU rd,rs1,imm`
Branches	Branch =	SB	`BEQ rs1,rs2,imm`
	Branch ≠	SB	`BNE rs1,rs2,imm`
	Branch <	SB	`BLT rs1,rs2,imm`
	Branch ≥	SB	`BGE rs1,rs2,imm`
	Branch < Unsigned	SB	`BLTU rs1,rs2,imm`
	Branch ≥ Unsigned	SB	`BGEU rs1,rs2,imm`
Jump & Link	J&L	UJ	`JAL rd,imm`
	Jump & Link Register	UJ	`JALR rd,rs1,imm`
Synch	Synch thread	I	`FENCE`
	Synch Instr & Data	I	`FENCE.I`
System	System CALL	I	`SCALL`
	System BREAK	I	`SBREAK`
Counters	ReaD CYCLE	I	`RDCYCLE rd`
	ReaD CYCLE upper Half	I	`RDCYCLEH rd`
	ReaD TIME	I	`RDTIME rd`
	ReaD TIME upper Half	I	`RDTIMEH rd`
	ReaD INSTR RETired	I	`RDINSTRET rd`
	ReaD INSTR upper Half	I	`RDINSTRETH rd`

Stores	Store Word	CS	`C.SW rs1',rs2',imm`		`SW rs1',rs2',imm*4`
	Store Word SP	CSS	`C.SWSP rs2,imm`		`SW rs2,sp,imm*4`
	Store Double	CS	`C.SD rs1',rs2',imm`		`SD rs1',rs2',imm*8`
	Store Double SP	CSS	`C.SDSP rs2,imm`		`SD rs2,sp,imm*8`
	Store Quad	CS	`C.SQ rs1',rs2',imm`		`SQ rs1',rs2',imm*16`
	Store Quad SP	CSS	`C.SQSP rs2,imm`		`SQ rs2,sp,imm*16`
Arithmetic	ADD	CR	`C.ADD`	`rd,rs1`	`ADD rd,rd,rs1`
	ADD Word	CR	`C.ADDW`	`rd,rs1`	`ADDW rd,rd,imm`
	ADD Immediate	CI	`C.ADDI`	`rd,imm`	`ADDI rd,rd,imm`
	ADD Word Imm	CI	`C.ADDIW`	`rd,imm`	`ADDIW rd,rd,imm`
	ADD SP Imm * 16	CI	`C.ADDI16SP`	`x0,imm`	`ADDI sp,sp,imm*16`
	ADD SP Imm * 4	CIW	`C.ADDI4SPN`	`rd',imm`	`ADDI rd',sp,imm*4`
	Load Immediate	CI	`C.LI`	`rd,imm`	`ADDI rd,x0,imm`
	Load Upper Imm	CI	`C.LUI`	`rd,imm`	`LUI rd,imm`
	MoVe	CR	`C.MV`	`rd,rs1`	`ADD rd,rs1,x0`
	SUB	CR	`C.SUB`	`rd,rs1`	`SUB rd,rd,rs1`
Shifts	Shift Left Imm	CI	`C.SLLI`	`rd,imm`	`SLLI rd,rd,imm`
Branches	Branch=0	CB	`C.BEQZ`	`rs1',imm`	`BEQ rs1',x0,imm`
	Branch≠0	CB	`C.BNEZ`	`rs1',imm`	`BNE rs1',x0,imm`
Jump	Jump	CJ	`C.J`	`imm`	`JAL x0,imm`
	Jump Register	CR	`C.JR`	`rd,rs1`	`JALR x0,rs1,0`
Jump & Link	J&L	CJ	`C.JAL`	`imm`	`JAL ra,imm`
	Jump & Link Register	CR	`C.JALR`	`rs1`	`JALR ra,rs1,0`
System	Env. BREAK	CI	`C.EBREAK`		`EBREAK`

https://www.cl.cam.ac.uk/teaching/1617/ECAD+Arch/files/docs/RISCVGreenCardv8-20151013.pdf

Ĉefprocesoro kaj traktadaj instrukcioj

Optional Multiply-Divide Instruction Extension: RVM						
Category	*Name*	*Fmt*	*RV32M (Multiply-Divide)*		*+RV{64,128}*	
Multiply	MULtiply	R	`MUL`	`rd,rs1,rs2`	`MUL{W|D}`	`rd,rs1,rs2`
	MULtiply upper Half	R	`MULH`	`rd,rs1,rs2`		
	MULtiply Half Sign/Uns	R	`MULHSU`	`rd,rs1,rs2`		
	MULtiply upper Half Uns	R	`MULHU`	`rd,rs1,rs2`		
Divide	DIVide	R	`DIV`	`rd,rs1,rs2`	`DIV{W|D}`	`rd,rs1,rs2`
	DIVide Unsigned	R	`DIVU`	`rd,rs1,rs2`		
Remainder	REMainder	R	`REM`	`rd,rs1,rs2`	`REM{W|D}`	`rd,rs1,rs2`
	REMainder Unsigned	R	`REMU`	`rd,rs1,rs2`	`REMU{W|D}`	`rd,rs1,rs2`
Optional Atomic Instruction Extension: RVA						
Category	*Name*	*Fmt*	*RV32A (Atomic)*		*+RV{64,128}*	
Load	Load Reserved	R	`LR.W`	`rd,rs1`	`LR.{D|Q}`	`rd,rs1`
Store	Store Conditional	R	`SC.W`	`rd,rs1,rs2`	`SC.{D|Q}`	`rd,rs1,rs2`
Swap	SWAP	R	`AMOSWAP.W`	`rd,rs1,rs2`	`AMOSWAP.{D|Q}`	`rd,rs1,rs2`
Add	ADD	R	`AMOADD.W`	`rd,rs1,rs2`	`AMOADD.{D|Q}`	`rd,rs1,rs2`
Logical	XOR	R	`AMOXOR.W`	`rd,rs1,rs2`	`AMOXOR.{D|Q}`	`rd,rs1,rs2`
	AND	R	`AMOAND.W`	`rd,rs1,rs2`	`AMOAND.{D|Q}`	`rd,rs1,rs2`
	OR	R	`AMOOR.W`	`rd,rs1,rs2`	`AMOOR.{D|Q}`	`rd,rs1,rs2`
Min/Max	MINimum	R	`AMOMIN.W`	`rd,rs1,rs2`	`AMOMIN.{D|Q}`	`rd,rs1,rs2`
	MAXimum	R	`AMOMAX.W`	`rd,rs1,rs2`	`AMOMAX.{D|Q}`	`rd,rs1,rs2`
	MINimum Unsigned	R	`AMOMINU.W`	`rd,rs1,rs2`	`AMOMINU.{D|Q}`	`rd,rs1,rs2`
	MAXimum Unsigned	R	`AMOMAXU.W`	`rd,rs1,rs2`	`AMOMAXU.{D|Q}`	`rd,rs1,rs2`
Three Optional Floating-Point Instruction Extensions: RVF, RVD, & RVQ						
Category	*Name*	*Fmt*	*RV32{F|D|Q} (HP/SP,DP,QP Fl Pt)*		*+RV{64,128}*	
Move	Move from Integer	R	`FMV.{H|S}.X`	`rd,rs1`	`FMV.{D|Q}.X`	`rd,rs1`
	Move to Integer	R	`FMV.X.{H|S}`	`rd,rs1`	`FMV.X.{D|Q}`	`rd,rs1`
Convert	Convert from Int	R	`FCVT.{H|S|D|Q}.W`	`rd,rs1`	`FCVT.{H|S|D|Q}.{L|T}`	`rd,rs1`
	Convert from Int Unsigned	R	`FCVT.{H|S|D|Q}.WU`	`rd,rs1`	`FCVT.{H|S|D|Q}.{L|T}U`	`rd,rs1`
	Convert to Int	R	`FCVT.W.{H|S|D|Q}`	`rd,rs1`	`FCVT.{L|T}.{H|S|D|Q}`	`rd,rs1`
	Convert to Int Unsigned	R	`FCVT.WU.{H|S|D|Q}`	`rd,rs1`	`FCVT.{L|T}U.{H|S|D|Q}`	`rd,rs1`

https://www.cl.cam.ac.uk/teaching/1617/ECAD+Arch/files/docs/RISCVGreenCardv8-20151013.pdf

Ĉefprocesoro kaj traktadaj instrukcioj

Load	Load	I	FL{W,D,Q}	rd,rs1,imm
Store	Store	S	FS{W,D,Q}	rs1,rs2,imm
Arithmetic	ADD	R	FADD.{S|D|Q}	rd,rs1,rs2
	SUBtract	R	FSUB.{S|D|Q}	rd,rs1,rs2
	MULtiply	R	FMUL.{S|D|Q}	rd,rs1,rs2
	DIVide	R	FDIV.{S|D|Q}	rd,rs1,rs2
	SQuare RooT	R	FSQRT.{S|D|Q}	rd,rs1
Mul-Add	Multiply-ADD	R	FMADD.{S|D|Q}	rd,rs1,rs2,rs3
	Multiply-SUBtract	R	FMSUB.{S|D|Q}	rd,rs1,rs2,rs3
	Negative Multiply-SUBtract	R	FNMSUB.{S|D|Q}	rd,rs1,rs2,rs3
	Negative Multiply-ADD	R	FNMADD.{S|D|Q}	rd,rs1,rs2,rs3
Sign Inject	SiGN source	R	FSGNJ.{S|D|Q}	rd,rs1,rs2
	Negative SiGN source	R	FSGNJN.{S|D|Q}	rd,rs1,rs2
	Xor SiGN source	R	FSGNJX.{S|D|Q}	rd,rs1,rs2
Min/Max	MINimum	R	FMIN.{S|D|Q}	rd,rs1,rs2
	MAXimum	R	FMAX.{S|D|Q}	rd,rs1,rs2
Compare	Compare Float =	R	FEQ.{S|D|Q}	rd,rs1,rs2
	Compare Float <	R	FLT.{S|D|Q}	rd,rs1,rs2
	Compare Float ≤	R	FLE.{S|D|Q}	rd,rs1,rs2
Categorization	Classify Type	R	FCLASS.{S|D|Q}	rd,rs1
Configuration	Read Status	R	FRCSR	rd
	Read Rounding Mode	R	FRRM	rd
	Read Flags	R	FRFLAGS	rd
	Swap Status Reg	R	FSCSR	rd,rs1
	Swap Rounding Mode	R	FSRM	rd,rs1
	Swap Flags	R	FSFLAGS	rd,rs1
	Swap Rounding Mode Imm	I	FSRMI	rd,imm
	Swap Flags Imm	I	FSFLAGSI	rd,imm

https://www.cl.cam.ac.uk/teaching/1617/ECAD+Arch/files/docs/RISCVGreenCardv8-20151013.pdf

Rapideco de komponentoj

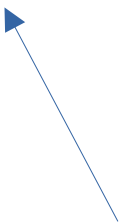
trakti instrukciono	2 ns	1 s
legi memoron de ĉefprocesoro	5 ns	2.5 s
legi ĉefmemoron	100 ns	50 s
legi diskon	1 000 000 ns	5.7 tagoj
sendu reta pakaĵo inter Eŭropo kaj Usono	150 000 000 ns	>2 jaroj

Konkludo

- oni ciferecigas signalojn por ricevi bitojn/bitokojn
- memoro estas vasta spaco de bitokoj
- klavaro/muso metas datumojn al memoro
  ekrano meta datumojn el memoro
- ĉefprocesoro procesas traktadajn instrukciojn
- oni verkas programojn kiel vico de instrukcioj

Dankon

Venonta klubvespero:
ĵau 2024-06-06
“La paradigmoj de programado”



Kiel oni mastrumas tiun vastan spacon de memoro?
Kiel oni modelas realajn problemojn?