Esperanta klubvespero

“La paradigmoj de programado”

prezentisto: Luko

ĵaudo la 2024-06-06

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

- Ekzemplo per RISC-V
- La funkciado de tradukilo kaj abstraktigo
- Paradigmoj
  - Ordonema paradigmo
  - Strukturema paradigmo
  - Objektema paradigmo
  - Funkciema paradigmo
- Vortprovizo kaj konkludo

01	06	FF	A0	B3	05	00	01	04	02	41	42	43
0x00000000	0x00000001	0x00000002	0x00000003	0x00000004	0x00000005							

bitoj / duuma (2)

0 1 0 0 0 0 0 1

signifo

A

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

deksesuma (16)

4 1

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

Traktadaj instrukcioj de ĉefprocesoro

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

Traktadaj instrukcioj de ĉefprocesoro

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

Traktadaj instrukcioj de ĉefprocesoro

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

Ekzemplo per RISC-V

Ekzemplo per RISC-V

```
.org 0x00
LW x1, 0x20(x0)
LW x2, 0x24(x0)
BLT x1, x2, 0x10
SW x1, 0xE0(x0)
EBREAK

.org 0x18
SW x2, 0xE0(x0)
EBREAK
```

```
.org 0x20
.word 0x42
.word 0x13

.org 0xE0
.word 00
```

Ekzemplo per RISC-V

programo

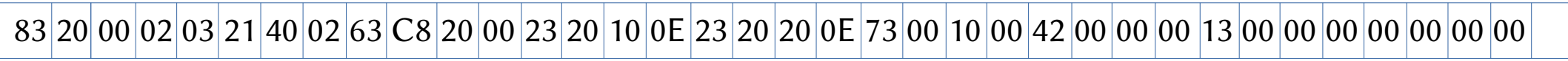
```
.org 0x00
LW x1, 0x20(x0)
LW x2, 0x24(x0)
BLT x1, x2, 0x10
SW x1, 0xE0(x0)
EBREAK

.org 0x18
SW x2, 0xE0(x0)
EBREAK
```

datumoj

```
.org 0x20
.word 0x42
.word 0x13

.org 0xE0
.word 00
```

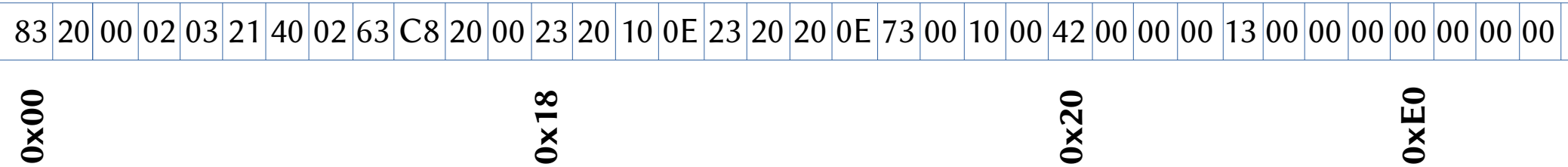



```
.org 0x00
LW x1, 0x20(x0)
LW x2, 0x24(x0)
BLT x1, x2, 0x10
SW x1, 0xE0(x0)
EBREAK

.org 0x18
SW x2, 0xE0(x0)
EBREAK
```

```
.org 0x20
.word 0x42
.word 0x13

.org 0xE0
.word 00
```

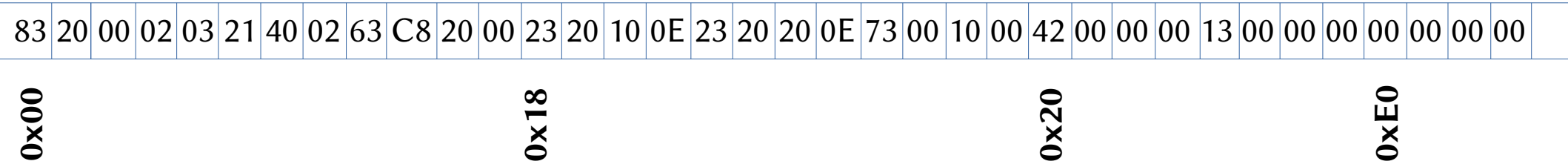



```
.org 0x00
LW x1, 0x20(x0)
LW x2, 0x24(x0)
BLT x1, x2, 0x10
SW x1, 0xE0(x0)
EBREAK

.org 0x18
SW x2, 0xE0(x0)
EBREAK
```

```
.org 0x20
.word 0x42
.word 0x13

.org 0xE0
.word 00
```

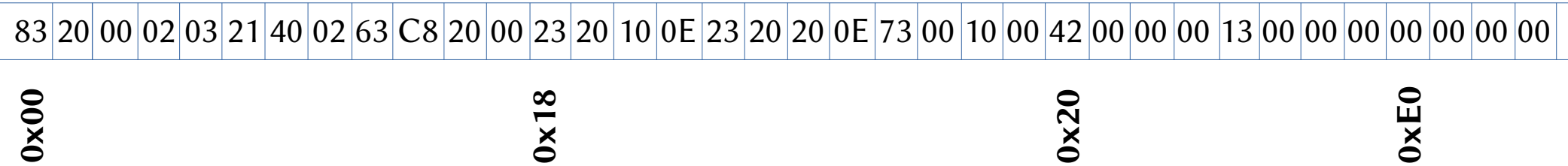



```
.org 0x00
LW x1, 0x20(x0)     x1 := 0x42
LW x2, 0x24(x0)
BLT x1, x2, 0x10
SW x1, 0xE0(x0)
EBREAK

.org 0x18
SW x2, 0xE0(x0)
EBREAK
```

```
.org 0x20
.word 0x42
.word 0x13

.org 0xE0
.word 00
```

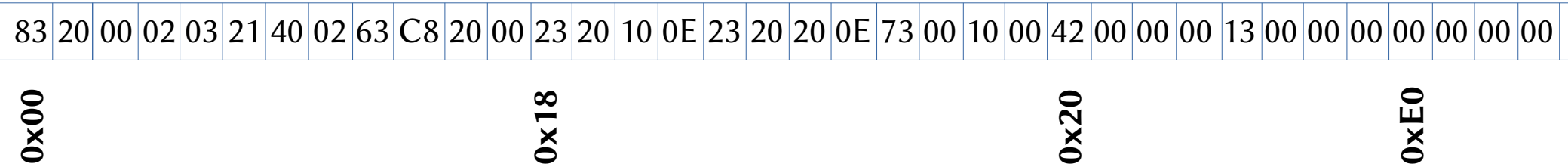



```
.org 0x00
LW x1, 0x20(x0)     x1 := 0x42
LW x2, 0x24(x0)     x2 := 0x13
BLT x1, x2, 0x10
SW x1, 0xE0(x0)
EBREAK

.org 0x18
SW x2, 0xE0(x0)
EBREAK
```

```
.org 0x20
.word 0x42
.word 0x13

.org 0xE0
.word 00
```

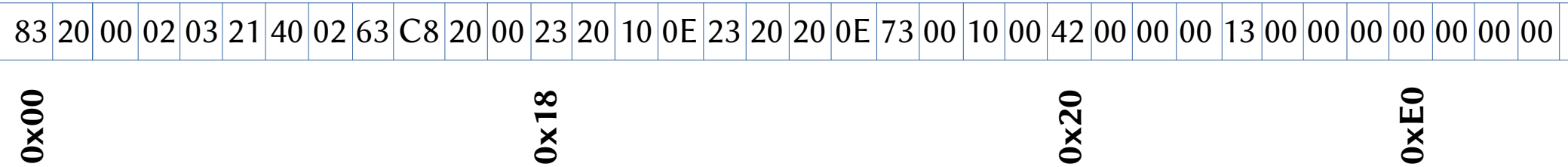



```
.org 0x00
LW x1, 0x20(x0)     x1 := 0x42
LW x2, 0x24(x0)     x2 := 0x13
BLT x1, x2, 0x10    se x1 < x2? Iru al +0x10
SW x1, 0xE0(x0)
EBREAK

.org 0x18
SW x2, 0xE0(x0)
EBREAK
```

```
.org 0x20
.word 0x42
.word 0x13

.org 0xE0
.word 00
```

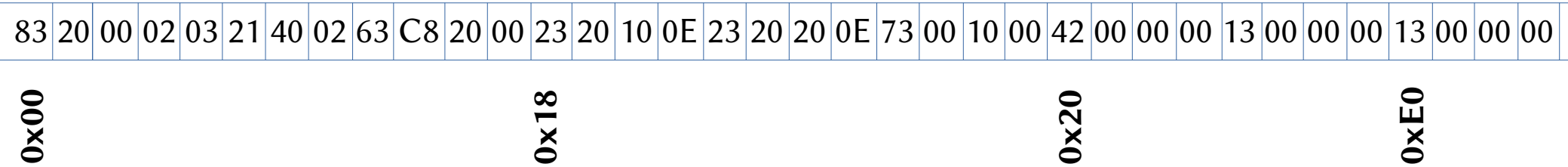



```
.org 0x00
LW x1, 0x20(x0)     x1 := 0x42
LW x2, 0x24(x0)     x2 := 0x13
BLT x1, x2, 0x10    se x1 < x2? Iru al +0x10
SW x1, 0xE0(x0)     če adreso 0xE0 := x1
EBREAK              haltu

.org 0x18
SW x2, 0xE0(x0)     če adreso 0xE0 := x2
EBREAK              haltu
```

```
.org 0x20
.word 0x42
.word 0x13

.org 0xE0
.word 00
```




```
.org 0x00
LW x1, 0x20(x0)     x1 := 0x42
LW x2, 0x24(x0)     x2 := 0x13
BLT x1, x2, 0x10    se x1 < x2? Iru al +0x10
SW x1, 0xE0(x0)     če adreso 0xE0 := x1
EBREAK              haltu

.org 0x18
SW x2, 0xE0(x0)     če adreso 0xE0 := x2
EBREAK              haltu
```

```
.org 0x20
.word 0x42
.word 0x13

.org 0xE0
.word 00
```

```
$ ./run example_01.asm

[…]
================================================================================
                        Final Simulation State
================================================================================
Processor State (e.g., Registers)
---------------------------------
PC: 00000018
00:00000000 01:00000042 02:00000013

RAM
---
83 20 00 02 03 21 40 02 63 C8 20 00 23 20 10 0E
23 20 20 0E 73 00 10 00 00 00 00 00 00 00 00 00
42 00 00 00 13 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00  00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
13 00 00 00

6 instructions simulated.
```

Alia programo

```
.org 0x00
LW x1, 0x30(x0)
ADDI x3, x0, 0x30

.org 0x08
ADDI x3, x3, 0x04
LW x2, 0x00(x3)
BEQ x2, x0, 0x10
BLT x1, x2, 0x08
ADDI x1, x2, 0x00
BEQ x0, x0, -0x14
```

```
.org 0x20
SW x1, 0xE0(x0)
EBREAK

.org 0x30
.word 0x42
.word 0x13
.word 0x0A
.org 0xE0
.word 0x00
```

Alia programo

```
.org 0x00
LW x1, 0x30(x0)
ADDI x3, x0, 0x30
```

x1 := 0x42

x3 := 0x30

```
.org 0x08
ADDI x3, x3, 0x04
LW x2, 0x00(x3)
BEQ x2, x0, 0x10
BLT x1, x2, 0x08
ADDI x1, x2, 0x00
BEQ x0, x0, -0x14
```

adiciu 4 al x3

x2 := nombro ĉe adr. en x3

se x2 egalas 0, iru al 0x20

x1 := x2

iru al 0x08

```
.org 0x20
SW x1, 0xE0(x0)
EBREAK
```

```
.org 0x30
.word 0x42
.word 0x13
.word 0x0A
.org 0xE0
.word 0x00
```

Alia programo

```
.org 0x00
LW x1, 0x30(x0)
ADDI x3, x0, 0x30

.org 0x08
ADDI x3, x3, 0x04
LW x2, 0x00(x3)
BEQ x2, x0, 0x10
BLT x1, x2, 0x08
ADDI x1, x2, 0x00
BEQ x0, x0, -0x14
```

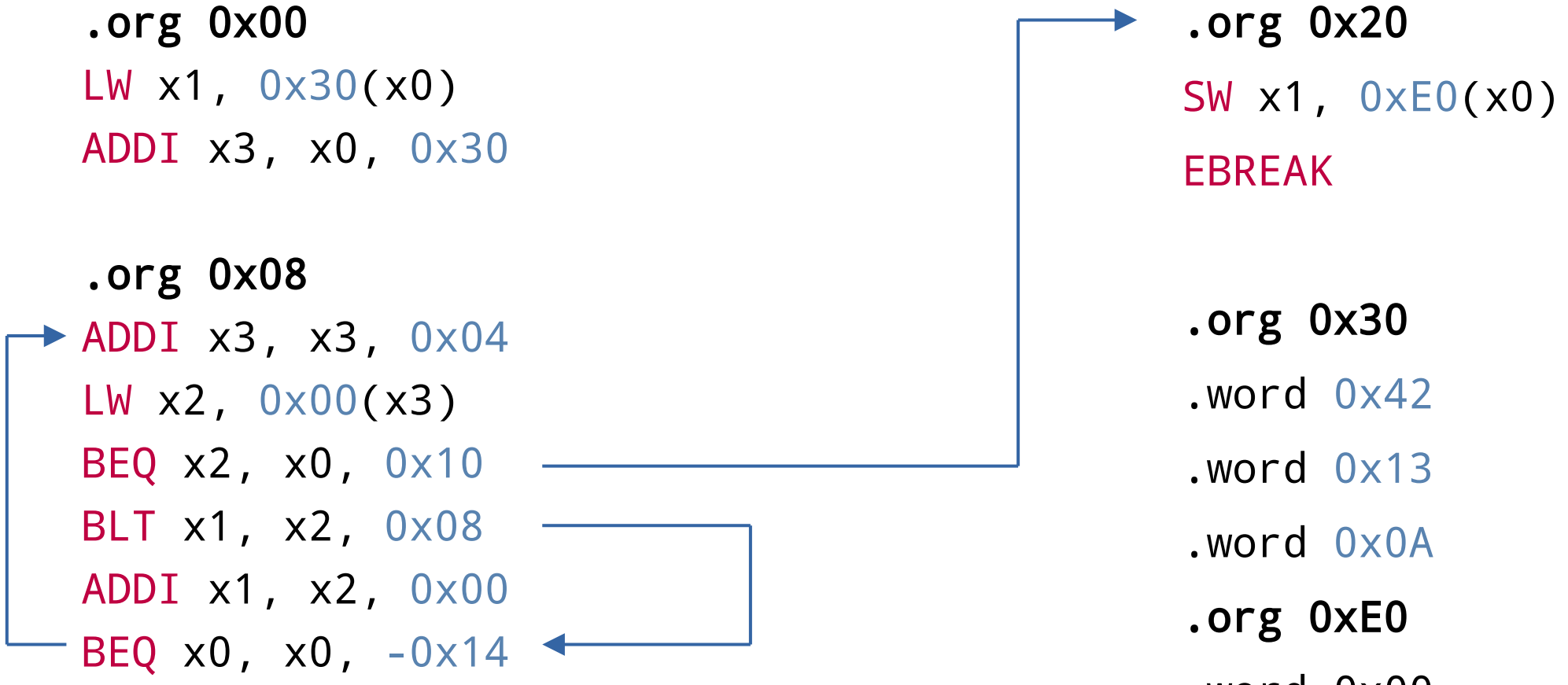
skribu x1 al adreso 0xE0

haltu

```
.org 0x20
SW x1, 0xE0(x0)
EBREAK

.org 0x30
.word 0x42
.word 0x13
.word 0x0A
.org 0xE0
.word 0x00
```

Alia programo



Kion vi lernis?

- **Algoritmo:** vico de paŝoj por solvi problemon
- **Asembla programlingvo:** lingvo por reprezenti trakteblajn paŝojn de ĉefprocesoro rekte

Ni diskutis algoritmojn por du problemoj:
1) Komputu la pli grandan nombron de du nombroj
2) Komputu la plej grandan nombron de vico

Ordonema paradigmo

- Paŝo post paŝo ni deskribas kiel oni atingas la celon
- Facilaj instrukcionoj, lineara progreso
- Ĉiu instrukciono havas saman gravon

Demando

Ĉu plaĉas al vi?

Tradukilo kaj abstraktigo

- Eble ni povas verki la programon en alia lingvo?
- Ni ordonas “iteracii” kaj ni povas kreii la instrukcionoj aŭtomate? (abstraktigo)
- Eble alia lingvo estas pli komprenebla?
- *Tradukilo* tradukas pli abstraktan programon al instrukcioj por komputilo.

Pliaj paradigmoj de programado

Strukturema programo (C)

```c
#include <stdio.h>

const int nombroj[] = {0x42, 0x13, 0x0A};

int main() {
    int minimumo = nombroj[0];

    int i;
    for (i = 0; i < 4; i++) {
        if (nombroj[i] < minimumo) {
            minimumo = nombroj[i];
        }
    }

    printf("minimumo = %02X\n", minimumo);      // minimumo = 0A
    return 0;
}
```

Strukturema programo

- Ni havas funkionojn, iteraciojn, kaj videblajn branĉojn
- Programo havas strukturojn en blokoj
- Libro “Structured programming” de Dahl, Dijkstra, kaj Hoare

Objektema paradigmo (Java)

```java
class Main {
  public static void main(String[] args) {
    Entjeroj ent = new Entjeroj();
    ent.aldonu(0x42);
    ent.aldonu(0x13);
    ent.aldonu(0x0A);
    System.out.println(
       String.format(
         "minimumo = %d",
         ent.komputu_minimumon()
       )
    );
  }
}
```

```java
import java.util.ArrayList;


class Entjeroj {
  ArrayList<Integer> entjeroj;

  public Entjeroj() {
    entjeroj = new ArrayList<Integer>();
  }

  public void aldonu(int nova_entjero) {
    this.entjeroj.add(
      new Integer(nova_entjero));
  }

  public int komputu_minimumon() {
    int kandidato = this.entjeroj.get(0);
    for (int nuna : this.entjeroj) {
      if (nuna < kandidato) {
        kandidato = nuna;
      }
    }
    return kandidato;
  }
}
```

Objektema paradigmo

- Ni havas objektojn kiu reprezentas kunajn datumojn sed ankaŭ kunan konduton
- Datumoj estas atributoj de la objekto.
  Konduto estas funkcioj (metodoj) de la objekto.
- Metodoj redaktas la atributojn.
- Pli: polimorfismo, heredado, enkapsulado
- Go kaj rust ne realigas la ideon de “subtyping”

Funkciema paradigmo (Haskell)

```haskell
import Data.List

nums = [0x42 :: Int, 0x13, 0x0A]

main = do print $ foldl (min) (head nums) (tail nums)
```

Funkciema paradigmo

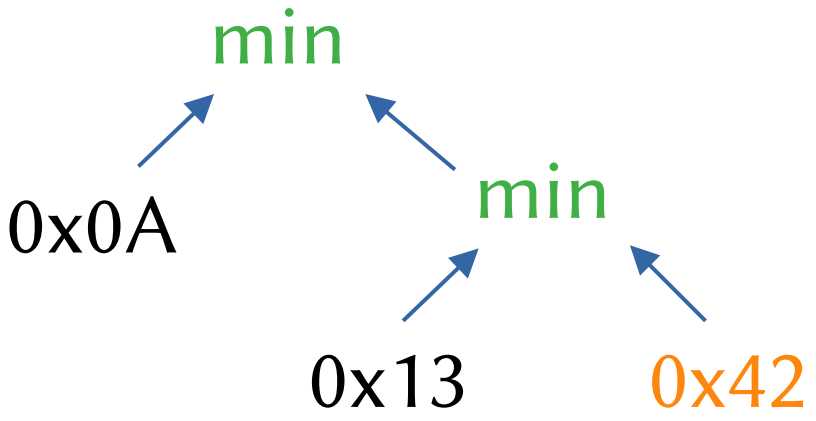
```
import Data.List

nums = [0x42 :: Int, 0x13, 0x0A]

main = do print $ foldl (min) (head nums) (tail nums)
```

0x42 [0x13, 0x0A]

foldl:



Funkciema paradigmo

- Funkcio estas tre grava parto. Funkcio agas kiel valoro (nombro/signoĉeno)
- Ni ne verkas programon paŝon post paŝon
  sed ni skribas kiuj funkcioj vokas kiujn funkcionojn
- Kiel matematiko: min(0x0A, min(0x13, 0x42))

La paradigmoj de programado

- Ekzistas pluraj programlingvoj (ASM, C, Java, Haskell)
- Paradigmoj agas kiel “kategorioj” de programlingvoj
- Laŭ mi, nun oni provas uzi pluraj programlingvoj samtempe en unu programlingvo (python, rust, ...)

Teorio de Montague

Richard Merritt Montague (1930–1971)

> “Laŭ mi ne ekzistas grava teoria diferenco inter **naturaj lingvoj kaj la artefaritaj lingvoj de logikistoj**; ja eblus ke oni povas kompreni la sintakson kaj semantikon de ambaŭ tiaj lingvoj per **sola natura kaj matematika akurata teorio**. Tiel mi kontraŭas kelkajn filozofistojn kaj kongruas – mi kredas – kun Chomsky et al”
>
> el “Universal Grammar” (1970)
>
> (emfazo kaj traduko de la angla Vikipedio de mi)

Teorio de Montague

category	symbol	type
Sentence	S	f
Verb phrase	VP	$t \to f$
Noun phrase	NP	$(t \to f) \to f$
Common noun	CN	$t \to f$
Determiner	DET	$(t \to f) \to ((t \to f) \to f)$
Transitive verb	TV	$t \to (t \to f)$

expression	meaning
a	$\lambda P.\, \lambda Q.\, \exists x((P\ x) \wedge (Q\ x))$
man	$\lambda x.\, MAN(x)$
a man	$\lambda Q.\, \exists x(MAN(x) \wedge (Q\ x))$
sleeps	$\lambda x.\, SLEEPS(x)$
a man sleeps	$\exists x(MAN(x) \wedge SLEEPS(x))$

		meaning
S	NP VP	$(NP\ VP)$
NP	name	$\lambda P.\, (P\ name)$
NP	DET CN	$(DET\ CN)$
NP	DET RCN	$(DET\ RCN)$
DET	"some"	$\lambda P.\, \lambda Q.\, \exists x((P\ x) \wedge (Q\ x))$
DET	"a"	$\lambda P.\, \lambda Q.\, \exists x((P\ x) \wedge (Q\ x))$
DET	"every"	$\lambda P.\, \lambda Q.\, \forall x((P\ x) \to (Q\ x))$
DET	"no"	$\lambda P.\, \lambda Q.\, \forall x((P\ x) \to \neg(Q\ x))$
VP	intransverb	$\lambda x.\, intransverb(x)$
VP	TV NP	$\lambda x.\, (NP\ \lambda y.\, (TV\ y\ x))$
TV	transverb	$\lambda y.\, \lambda x.\, transverb(x, y)$
RCN	CN "that" VP	$\lambda x.\, ((CN\ x) \wedge (VP\ x))$
RCN	CN "that" NP TV	$\lambda x.\, ((CN\ x) \wedge (NP\ \lambda y.\, (TV\ y\ x)))$
CN	predicate	$\lambda x.\, predicate(x)$

Vortprovizo

- algoritmo = algorithm
- asembla programlingvo = assembly language
- paradigmoj de programado = software paradigms
- ordonema/objektema/funkciema = imperative/object-oriented/functional
- abstraktigo = abstraction
- tradukilo = compiler

Konkludo

- Programistoj serĉas por programlingvo
  kiu prezentas la solvon kiel la solvo en nia cerbo
- Paradigmoj estas kategorioj de programlingvoj
- Krei planlingvojn similas al krei programlingvojn

Dankon

Venonta klubvespero:
ĵau 2024-07-04
“La programlingvo trankvila”

de Rikardo