MiMo - Mikroprogramska shema kontrolne enote v0.4

Naslov/ signal	Kontrolni (»Control«) ROM 256x32bitov (23 izkoriščenih)														Opis vsebine mikroprograma			Odločitveni (»Decision«) ROM 256x16bitov	
	1	2	1	2	2	1	1	1	1	2	2	1	2	4					
	swrite	datasel	indexsel	cond	regsrc	imload	irload	dwrite	pcload	pcsel	addrsel	datawrite	op2sel	aluop	Oznaka/ op.koda:	Ukaz, Mikroukaz	Opis operacije, mikroukaza	true 8bit	false 8bit
0							1				0				fetch:	IR<-M[PC]	IR<-M[PC], goto [1]	1	1
1			1						1	0						PC<-PC+1	PC<-PC+1, goto »Opcode+2«	2	2
2					2			1							0:	ADD Rd,Rs,Rt	ALU ADD oper.on Rd,Rs,Rt, goto fetch:	0	0
42 0x2a						1					0				40:	JNEZ Rs,immed	immed<-M[PC], goto [0x82]	82	82
65 0x41								1			0				63:	LI Rd,Immed	Rd <-M[PC] goto pcincr:	84	84
67 0x43						1					0				65:	SW Rd,immed	immed<-M[PC], goto [0x83]	83	83
130 0x82				2									2	1		JNEZ Rs,immed	ALU:SUB Rs-0, if Zero then pcincr: else jump:	84	85
131 0x83		1									1	1				SW Rd,immed	Addr=immed ,Rd->M[immed]; goto pcincr:	84	84
132 0x84									1						pcincr:	PC++, goto fetch:	PC<-PC+1, goto fetch:	0	0
133 0x85									1	1					jump:	PC<-immed, goto fetch:	immed->PC, Addr=immed, goto fetch:	0	0

datasel:
0..PC
1..Dreg
2..Treg
3..ALU

regsrc:
0..DBus
1..IMM
2..ALU
3..Sreg

pcsel:
0..PC+1
1..IMM
2..PC+IMM
3..Sreg

addrsel:
0..PC
1..IMM
2..ALU
3..Sreg

op2sel:
0..Treg
1..IMM
2.."0"
3.."1"

aluop:
0..+
1..-
2..*
3../
...

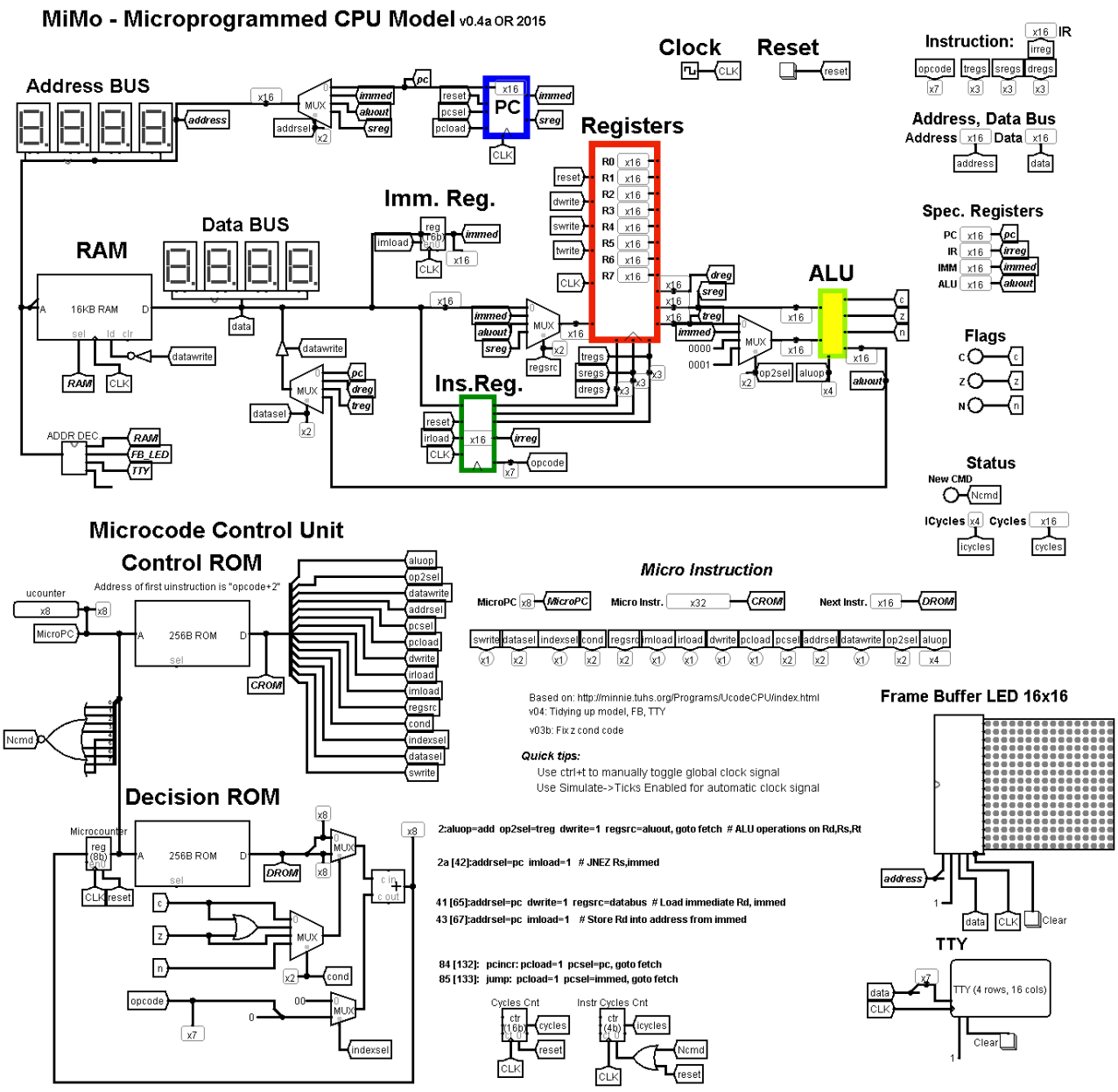
cond:
0..C
1..C or Z
2..Z
3..N

Format 1:

Op.koda	Treg	Sreg	Dreg
7	3	3	3

Format 2:
Format 1 + 16-bitni tak. operand

v 0.4


MiMo - Microprogrammed CPU Model v0.4a OR 2015
Clock
Reset
Instruction:
IR
Address BUS
Registers
Address, Data Bus
Address
Data
Imm. Reg.
Data BUS
Spec. Registers
RAM
ALU
Flags
Ins.Reg.
Status
New CMD
ICycles
Cycles
Microcode Control Unit
Control ROM
Address of first uinstruction is "opcode+2"
Micro Instruction
Decision ROM
Based on: http://minnie.tuhs.org/Programs/UcodeCPU/index.html
v04: Tidying up model, FB, TTY
v03b: Fix z cond code
Quick tips:
Use ctrl+t to manually toggle global clock signal
Use Simulate->Ticks Enabled for automatic clock signal
2:aluop=add op2sel=treg dwrite=1 regsrc=aluout, goto fetch # ALU operations on Rd,Rs,Rt
2a [42]:addrsel=pc imload=1 # JNEZ Rs,immed
41 [65]:addrsel=pc dwrite=1 regsrc=databus # Load immediate Rd, immed
43 [67]:addrsel=pc imload=1 # Store Rd into address from immed
84 [132]: pcincr: pcload=1 pcsel=pc, goto fetch
85 [133]: jump: pcload=1 pcsel=immed, goto fetch
Frame Buffer LED 16x16
TTY
TTY (4 rows, 16 cols)

Spisek in opis podprtih ukazov v zbirniku

add Rd,Rs,Rt (0)
Rd <- Rs + Rt PC <- PC + 1

sub Rd,Rs,Rt (1)
Rd <- Rs - Rt PC <- PC + 1

mul Rd,Rs,Rt (2)
Rd <- Rs * Rt PC <- PC + 1

div Rd,Rs,Rt (3)
Rd <- Rs / Rt PC <- PC + 1

rem Rd,Rs,Rt (4)
Rd <- Rs % Rt PC <- PC + 1

and Rd,Rs,Rt (5)
Rd <- Rs AND Rt PC <- PC + 1

or Rd,Rs,Rt (6)
Rd <- Rs OR Rt PC <- PC + 1

xor Rd,Rs,Rt (7)
Rd <- Rs XOR Rt PC <- PC + 1

nand Rd,Rs,Rt (8)
Rd <- Rs NAND Rt PC <- PC + 1

nor Rd,Rs,Rt (9)
Rd <- Rs NOR Rt PC <- PC + 1

not Rd,Rs (10)
Rd <- NOT Rs PC <- PC + 1

lsl Rd,Rs,Rt (11)
Rd <- Rs << Rt PC <- PC + 1

lsr Rd,Rs,Rt (12)
Rd <- Rs >> Rt PC <- PC + 1

asr Rd,Rs,Rt (13)
Rd <- Rs >> Rt (filled bits are the sign bit) PC <- PC + 1

rol Rd,Rs,Rt (14)
Rd <- Rs rolled left by Rt bits PC <- PC + 1

ror Rd,Rs,Rt (15)
Rd <- Rs rolled right by Rt bits PC <- PC + 1

addi Rd,Rs,immed (16)
Rd <- Rs + immed PC <- PC + 2

subi Rd,Rs,immed (17)
Rd <- Rs - immed PC <- PC + 2

muli Rd,Rs,immed (18)
Rd <- Rs * immed PC <- PC + 2

divi Rd,Rs,immed (19)
Rd <- Rs / immed PC <- PC + 2

remi Rd,Rs,immed (20)
Rd <- Rs % immed PC <- PC + 2

andi Rd,Rs,immed (21)
Rd <- Rs AND immed PC <- PC + 2

ori Rd,Rs,immed (22)
Rd <- Rs OR immed PC <- PC + 2

xori Rd,Rs,immed (23)
Rd <- Rs XOR immed PC <- PC + 2

nandi Rd,Rs,immed (24)
Rd <- Rs NAND immed PC <- PC + 2

nori Rd,Rs,immed (25)
Rd <- Rs NOR immed PC <- PC + 2

lsli Rd,Rs,immed (26)
Rd <- Rs << immed PC <- PC + 2

lsri Rd,Rs,immed (27)
Rd <- Rs >> immed PC <- PC + 2

asri Rd,Rs,immed (28)
Rd <- Rs >> immed (filled bits are the sign bit) PC <- PC + 2

roli Rd,Rs,immed (29)
Rd <- Rs rolled left by immed bits PC <- PC + 2

rori Rd,Rs,immed (30)
Rd <- Rs rolled right by immed bits PC <- PC + 2

addc Rd,Rs,Rt,immed (31)
Rd <- Rs + Rt
if carry set, PC <- immed else PC <- PC + 2

subc Rd,Rs,Rt,immed (32)
Rd <- Rs - Rt
if carry set, PC <- immed else PC <- PC + 2

jeq Rs,Rt,immed (33)
if Rs == Rt, PC <- immed else PC <- PC + 2

jne Rs,Rt,immed (34)
if Rs != Rt, PC <- immed else PC <- PC + 2

jgt Rs,Rt,immed (35)
if Rs > Rt, PC <- immed else PC <- PC + 2

jle Rs,Rt,immed (36)
if Rs <= Rt, PC <- immed else PC <- PC + 2

jlt Rs,Rt,immed (37)
if Rs < Rt, PC <- immed else PC <- PC + 2

jge Rs,Rt,immed (38)
if Rs >= Rt, PC <- immed else PC <- PC + 2

jeqz Rs,immed (39)
if Rs == 0, PC <- immed else PC <- PC + 2

jnez Rs,immed (40)
if Rs != 0, PC <- immed else PC <- PC + 2

jgtz Rs,immed (41)
if Rs > 0, PC <- immed else PC <- PC + 2

jlez Rs,immed (42)
if Rs <= 0, PC <- immed else PC <- PC + 2

jltz Rs,immed (43)
if Rs < 0, PC <- immed else PC <- PC + 2

jgez Rs,immed (44)
if Rs >= 0, PC <- immed else PC <- PC + 2

jmp immed (45)
PC <- immed

beq Rs,Rt,immed (46)
if Rs == Rt, PC <- PC + immed else PC <- PC + 2

bne Rs,Rt,immed (47)
if Rs != Rt, PC <- PC + immed else PC <- PC + 2

bgt Rs,Rt,immed (48)
if Rs > Rt, PC <- PC + immed else PC <- PC + 2

ble Rs,Rt,immed (49)
if Rs <= Rt, PC <- PC + immed else PC <- PC + 2

blt Rs,Rt,immed (50)
if Rs < Rt, PC <- PC + immed else PC <- PC + 2

bge Rs,Rt,immed (51)
if Rs >= Rt, PC <- PC + immed else PC <- PC + 2

beqz Rs,immed (52)
if Rs == 0, PC <- PC + immed else PC <- PC + 2

bnez Rs,immed (53)
if Rs != 0, PC <- PC + immed else PC <- PC + 2

bgtz Rs,immed (54)
if Rs > 0, PC <- PC + immed else PC <- PC + 2

blez Rs,immed (55)
if Rs <= 0, PC <- PC + immed else PC <- PC + 2

bltz Rs,immed (56)
if Rs < 0, PC <- PC + immed else PC <- PC + 2

bgez Rs,immed (57)
if Rs >= 0, PC <- PC + immed else PC <- PC + 2

br immed (58)
PC <- PC + immed

Register 7 is used as the stack pointer. It points at the most-recently
pushed value on the stack. M[ ] means the memory cell at the location
in the brackets.

jsr immed (59)
R7--
M[R7] <- PC + 2, i.e. skip the current 2-word instruction
PC <- immed

rts (60)
PC <- M[R7]
R7++

inc Rs (61)
Rs <- Rs + 1 PC <- PC + 1

dec Rs (62)
Rs <- Rs - 1 PC <- PC + 1

li Rd,immed (63)
Rd <- immed PC <- PC + 2

lw Rd,immed (64)
Rd <- M[immed] PC <- PC + 2

sw Rd,immed (65)
M[immed] <- Rd PC <- PC + 2

lwi Rd,Rs,immed (66)
Rd <- M[Rs+immed] PC <- PC + 2

swi Rd,Rs,immed (67)
M[Rs+immed] <- Rd PC <- PC + 2

push Rd (68)
R7--
M[R7] <- Rd PC <- PC + 1

pop Rd (69)
Rd <- M[R7]
R7++ PC <- PC + 1

move Rd,Rs (70)
Rd <- Rs PC <- PC + 1

clr Rs (71)
Rs <- 0 PC <- PC + 1

neg Rs (72)
Rs <- -Rs PC <- PC + 1

lwri Rd,Rs,Rt (73)
Rd <- M[Rs+Rt] PC <- PC + 1

swri Rd,Rs,Rt (74)
M[Rs+Rt] <- Rd PC <- PC + 1