Computer Architecture

Simulators

Course: 361-1-4201

Dr. Guy Tel-Zur

במצגת זו מתוארים כלי סימולציה, חופשיים, אשר יכולים לסייע בהבנה של חומרי הלימוד באמצעות תרגול עצמי.

המצגת תתעדכן מעת לעת. עדכון אחרון: 28/5/2023

In 3 Parts...

Part 1: MIPS

Part 2: RISC-V

Part 3: Advanced topics

Part 1: MIPS

MARS

Java based MIPS simulator

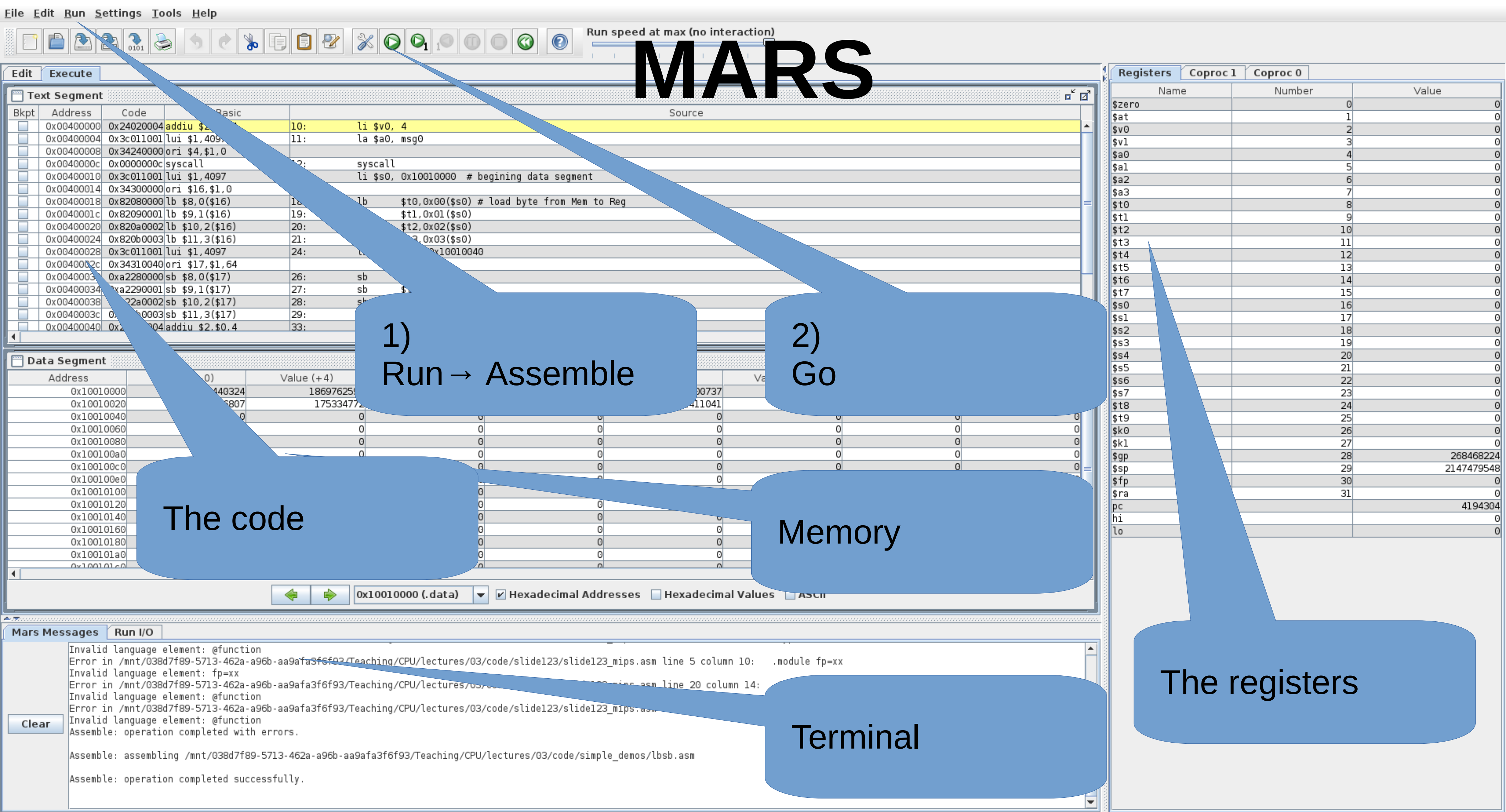
Home page:
http://courses.missouristate.edu/KenVollmar/MARS/

MARS

Invocation (Linux):

```
File  Edit  View  Search  Terminal  Help
telzur@GL553VD ~ $ which mars
/home/telzur/bin/mars
telzur@GL553VD ~ $ cat ~/bin/mars
#!/bin/sh
java -jar /home/telzur/science/Teaching/CPU/SW/MARS/Mars4_5.jar
telzur@GL553VD ~ $
```

MARS



```
Bkpt Address    Code       Basic                    Source
☐    0x00400000 0x24020004 addiu $2               10:        li $v0, 4
☐    0x00400004 0x3c011001 lui $1,409             11:        la $a0, msg0
☐    0x00400008 0x34240000 ori $4,$1,0
☐    0x0040000c 0x0000000c syscall                  :        syscall
☐    0x00400010 0x3c011001 lui $1,4097                       li $s0, 0x10010000  # begining data segment
☐    0x00400014 0x34300000 ori $16,$1,0
☐    0x00400018 0x82080000 lb $8,0($16)           1          lb      $t0,0x00($s0) # load byte from Mem to Reg
☐    0x0040001c 0x82090001 lb $9,1($16)           19:                $t1,0x01($s0)
☐    0x00400020 0x820a0002 lb $10,2($16)          20:                $t2,0x02($s0)
☐    0x00400024 0x820b0003 lb $11,3($16)          21:                   ,0x03($s0)
☐    0x00400028 0x3c011001 lui $1,4097            24:                   x10010040
☐    0x0040002c 0x34310040 ori $17,$1,64
☐    0x0040003  0xa2280000 sb $8,0($17)           26:        sb
☐    0x00400034  xa2290001 sb $9,1($17)           27:        sb      $
☐    0x00400038    22a0002 sb $10,2($17)          28:        s
☐    0x0040003c 0    b0003 sb $11,3($17)          29:
☐    0x00400040 0x2    004 addiu $2,$0,4          33:
```

Name	Number	Value
$zero	0	0
$at	1	0
$v0	2	0
$v1	3	0
$a0	4	0
$a1	5	0
$a2	6	0
$a3	7	0
$t0	8	0
$t1	9	0
$t2	10	0
$t3	11	0
$t4	12	0
$t5	13	0
$t6	14	0
$t7	15	0
$s0	16	0
$s1	17	0
$s2	18	0
$s3	19	0
$s4	20	0
$s5	21	0
$s6	22	0
$s7	23	0
$t8	24	0
$t9	25	0
$k0	26	0
$k1	27	0
$gp	28	268468224
$sp	29	2147479548
$fp	30	0
$ra	31	0
pc		4194304
hi		0
lo		0

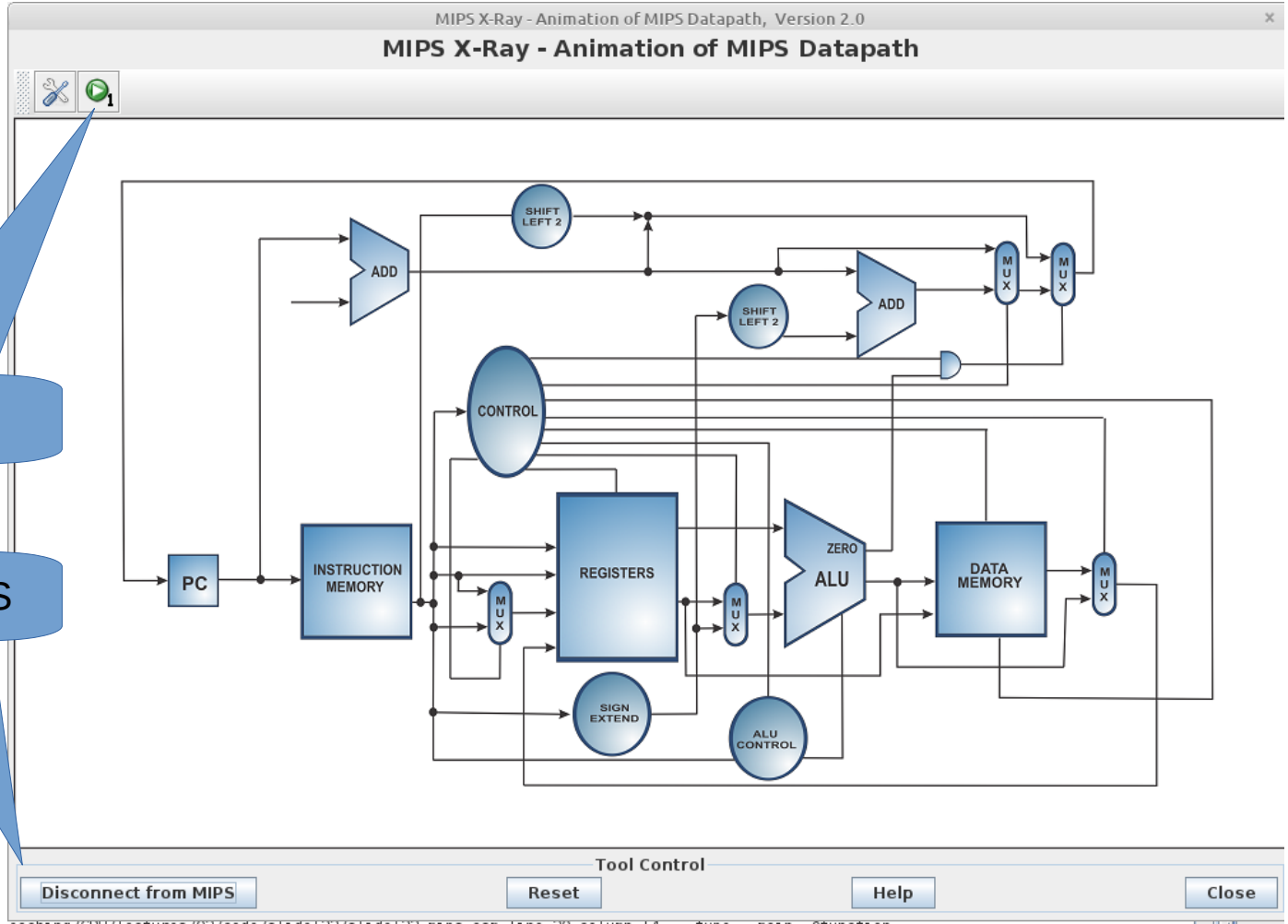
```
Invalid language element: @function
Error in /mnt/038d7f89-5713-462a-a96b-aa9afa3f6f93/Teaching/CPU/lectures/03/code/slide123/slide123_mips.asm line 5 column 10:   .module fp=xx
Invalid language element: fp=xx
Error in /mnt/038d7f89-5713-462a-a96b-aa9afa3f6f93/Teaching/CPU/lectures/03/                  mips.asm line 20 column 14:
Invalid language element: @function
Error in /mnt/038d7f89-5713-462a-a96b-aa9afa3f6f93/Teaching/CPU/lectures/03/code/slide123/slide123_mips.a
Invalid language element: @function
Assemble: operation completed with errors.

Assemble: assembling /mnt/038d7f89-5713-462a-a96b-aa9afa3f6f93/Teaching/CPU/lectures/03/code/simple_demos/lbsb.asm

Assemble: operation completed successfully.
```

MARS

Tools → X-Ray

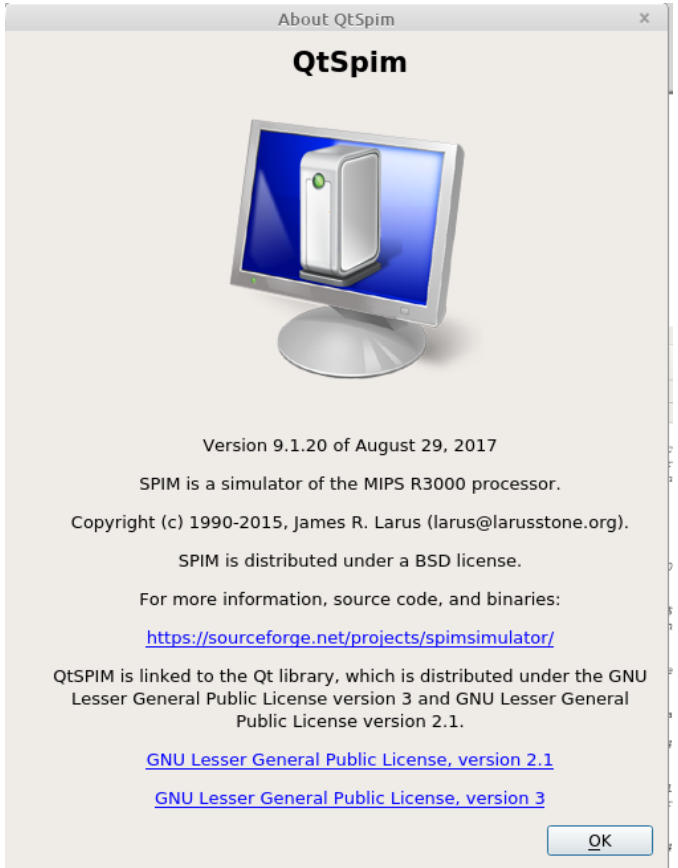
2) Execute

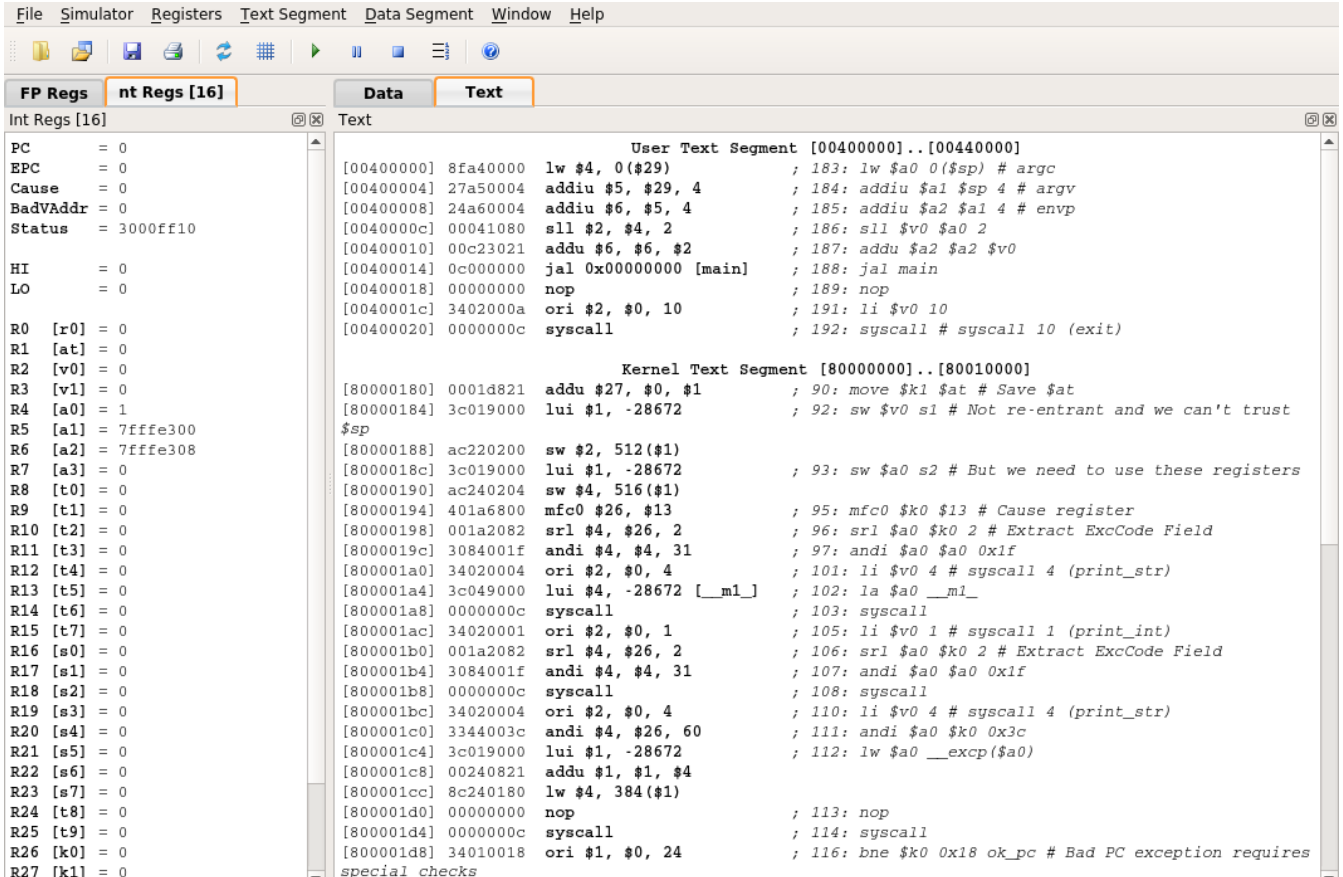
1) Connect to MIPS



SPIM / QtSPIM

https://spimsimulator.sourceforge.net/





QtSPIM

The code is loaded into the simulator. To execute click on “Simulator” → Run (F5) or Single Step (F10)

```
telzur@GL553VD ~/tmp/RISC                    e ./qtspim_example.s
     .text
     .globl main
main:
     li $t0, 0x2 # $t0  0x2
     li $t1, 0x3 # $t1  0x3
     addu $t2, $t0, $t1 # $t2  ADD($t0, $t1)
telzur@GL553VD ~/tmp/RISC-V $
```

QtSpim

File Simulator Registers Text Segment Data Segment Window Help

FP Regs | nt Regs [16]

Int Regs [16]

```
PC       = 400030
EPC      = 0
Cause    = 0
BadVAddr = 0
Status   = 3000ff10

HI       = 0
LO       = 0

R0  [r0] = 0
R1  [at] = 0
R2  [v0
R3
                 1ffe320
                 7fffe328
                 = 0
                 = 2
                 = 3
                 = 5
                 3] = 0
                 [t4] = 0
                 3 [t5] = 0
R25 [t9] = 0
R26 [k0] = 0
R27 [k1] = 0
```

```
                    User Text Segment [00400000]..[00440000]
[00400000] 8fa40000  lw $4, 0($29)              ; 183: lw $a0 0($sp) # argc
[00400004] 27a50004  addiu $5, $29, 4           ; 184: addiu $a1 $sp 4 # argv
[00400008] 24a60004  addiu $6, $5, 4            ; 185: addiu $a2 $a1 4 # envp
[0040000c] 00041080  sll $2, $4, 2              ; 186: sll $v0 $a0 2
[00400010] 00c23021  addu $6, $6, $2            ; 187: addu $a2 $a2 $v0
[00400014] 0c100009  jal 0x00400024 [main]      ; 188: jal main
[00400018] 00000000  nop                        ; 189: nop
[0040001c] 3402000a  ori $2, $0, 10             ; 191: li $v0 10
[00400020] 0000000c  syscall                    ; 192: syscall # syscall 10 (exit)
[00400024] 34080002  ori $8, $0, 2              ; 4: li $t0, 0x2 # $t0 0x2
[00400028] 34090003  ori $9, $0, 3              ; 5: li $t1, 0x3 # $t1 0x3
[0040002c] 01095021  addu $10, $8, $9           ; 6: addu $t2, $t0, $t1 # $t2 ADD($t0, $t1)

                    Kernel Text Segment [80000000]..[80010000]
[80000180] 0001d821  addu $27, $0, $1           ; 90: move $k1 $at # Save $at
[80000184] 3c019000  lui $1, -28672             ; 92: sw $v0 s1 # Not re-entrant and we can't trust
$sp
[80000188] ac220200  sw $2, 512($1)
[8000018c] 3c019000  lui $1, -28672             ; 93: sw $a0 s2 # But we need to use these registers
[80000190] ac240204  sw $4, 516($1)
[80000194] 401a6800  mfc0 $26, $13              ; 95: mfc0 $k0 $13 # Cause register
[80000198] 001a2082  srl $4, $26, 2             ; 96: srl $a0 $k0 2 # Extract ExcCode Field
[8000019c] 3084001f  andi $4, $4, 31            ; 97: andi $a0 $a0 0x1f
[800001a0] 34020004  ori $2, $0, 4              ; 101: li $v0 4 # syscall 4 (print_str)
[800001a4] 3c049000  lui $4, -28672 [__m1_]     ; 102: la $a0 __m1_
[800001a8] 0000000c  syscall                    ; 103: syscall
[800001ac] 34020001  ori $2, $0, 1              ; 105: li $v0 1 # syscall 1 (print_int)
[800001b0] 001a2082  srl $4, $26, 2             ; 106: srl $a0 $k0 2 # Extract ExcCode Field
[800001b4] 3084001f  andi $4, $4, 31            ; 107: andi $a0 $a0 0x1f
[800001b8] 0000000c  syscall                    ; 108: syscall
[800001bc] 34020004  ori $2, $0, 4              ; 110: li $v0 4 # syscall 4 (print_str)
[800001c0] 3344003c  andi $4, $26, 60           ; 111: andi $a0 $k0 0x3c
[800001c4] 3c019000  lui $1, -28672             ; 112: lw $a0 __excp($a0)
[800001c8] 00240821  addu $1, $1, $4
[800001cc] 8c240180  lw $4, 384($1)
[800001d0] 00000000  nop                        ; 113: nop
```

```
Attempt to execute non-instruction at 0x00400030
Attempt to execute non-instruction at 0x00400030
Attempt to execute non-instruction at 0x00400030
Attempt to execute non-instruction at 0x00400030
```

EduMIPS

https://edumips.org/

EduMIPS64 v. 1.2.7.1 - MIPS64 Processor Simulator

File Execute Configure Tools Window Help

Cycles

Home screen

Registers

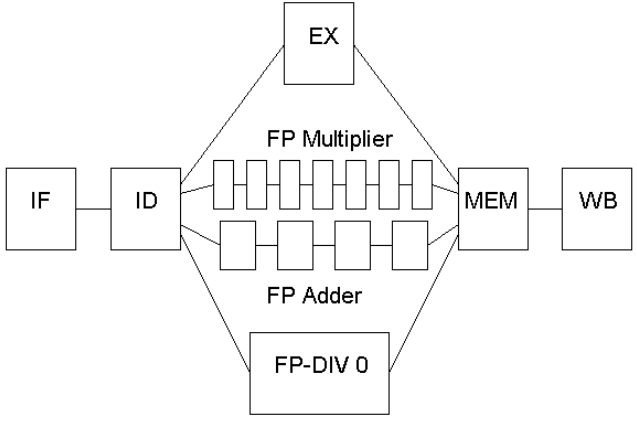
R0	($zero)	0000000000000000	F0	000000000000000
R1	($at)	0000000000000000	F1	000000000000000
R2	($v0)	0000000000000000	F2	000000000000000
R3	($v1)	0000000000000000	F3	000000000000000
R4	($a0)	0000000000000000	F4	000000000000000
R5	($a1)	0000000000000000	F5	000000000000000
R6	($a2)	0000000000000000	F6	000000000000000
R7	($a3)	0000000000000000	F7	000000000000000
R8	($t0)	0000000000000000	F8	000000000000000
R9	($t1)	0000000000000000	F9	000000000000000
R10	($t2)	0000000000000000	F10	000000000000000
R11	($t3)	0000000000000000	F11	000000000000000
R12	($t4)	0000000000000000	F12	000000000000000
R13	($t5)	0000000000000000	F13	000000000000000
R14	($t6)	0000000000000000	F14	000000000000000
R15	($t7)	0000000000000000	F15	000000000000000
R16	($s0)	0000000000000000	F16	000000000000000
R17	($s1)	0000000000000000	F17	000000000000000

Statistics

Execution
0 Cycles
0 Instructions

Stalls
0 RAW Stalls
0 WAW Stalls
0 WAR Stalls
0 Structural Stalls (Divider not available)
0 Structural Stalls (Memory not available)
0 Branch Taken Stalls
0 Branch Misprediction Stalls
Code size
0 Bytes
Floating point unit
FCSR register
FCC Cause EnablFlag RM
7654321 0 VZOUIVZOUIVZOUI
00000000000000000000111100000001

Pipeline



Data

Address	Representation	Label	Data
0000	0000000000000000		
0008	0000000000000000		
0010	0000000000000000		
0018	0000000000000000		
0020	0000000000000000		
0028	0000000000000000		
0030	0000000000000000		
0038	0000000000000000		
0040	0000000000000000		
0048	0000000000000000		
0050	0000000000000000		
0058	0000000000000000		
0060	0000000000000000		
0068	0000000000000000		
0070	0000000000000000		
0078	0000000000000000		
0080	0000000000000000		

Code

Address	Representation	Label	Instruction
0000			
0004			
0008			
000C			
0010			
0014			
0018			
001C			
0020			
0024			
0028			
002C			
0030			
0034			
0038			
003C			
0040			

Welcome to EduMIPS64 1.2.7.1

EduMIPS64 v. 1.2.7.1 - MIPS64 Processor Simulator - guy1.asm

File Execute Configure Tools Window Help

Cycles

daddi r5,r0,format_	IF	ID	EX	MEM	WB
sw r5,fs_addr(r0)		IF	ID	RAW	RAW
daddi r2,r0,s1			IF		

While running a code

Registers

R0	($zero)	0000000000000000	F0	000000000000000
R1	($at)	0000000000000000	F1	000000000000000
R2	($v0)	0000000000000000	F2	000000000000000
R3	($v1)	0000000000000000	F3	000000000000000
R4	($a0)	0000000000000000	F4	000000000000000
R5	($a1)	0000000000000000	F5	000000000000000
R6	($a2)	0000000000000000	F6	000000000000000
R7	($a3)	0000000000000000	F7	000000000000000
R8	($t0)	0000000000000000	F8	000000000000000
R9	($t1)	0000000000000000	F9	000000000000000
R10	($t2)	0000000000000000	F10	000000000000000
R11	($t3)	0000000000000000	F11	000000000000000
R12	($t4)	0000000000000000	F12	000000000000000
R13	($t5)	0000000000000000	F13	000000000000000
R14	($t6)	0000000000000000	F14	000000000000000
R15	($t7)	0000000000000000	F15	000000000000000
R16	($s0)	0000000000000000	F16	000000000000000
R17	($s1)	0000000000000000	F17	000000000000000

Statistics

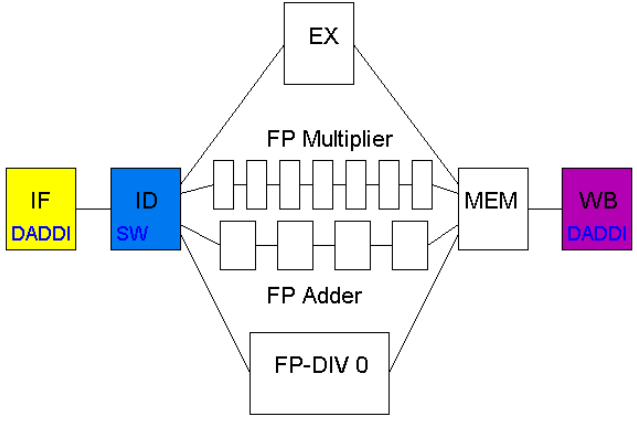
Execution
5 Cycles
0 Instructions

Stalls
2 RAW Stalls
0 WAW Stalls
0 WAR Stalls
0 Structural Stalls (Divider not available)
0 Structural Stalls (Memory not available)
0 Branch Taken Stalls
0 Branch Misprediction Stalls

Code size
36 Bytes

Floating point unit
FCSR register
FCC Cause EnablFlag RM
7654321 0 VZOUIVZOUIVZOUI
00000000000000000000111100000001

Pipeline



Data

Address	Representation	Label	Data
0000	20666F2068746425	format_str	.asciiz "%dth of %s
0008	762073250A3A7325		
0010	25206E6F69737265		
0018	2073692069252E69		
0020	657420676E696562		
0028	0000002164657473		
0030	00000000656E754A	s1	.asciiz "June"
0038	365350494D756445	s2	.asciiz "EduMIPS64"
0040	0000000000000034		
0048	0000000000000000	fs_addr	.space 4
0050	0000000000000005		.word 5
0058	0000000000000000	s1_addr	.space 4
0060	0000000000000000	s2_addr	.space 4
0068	0000000000000000		.word 0
0070	0000000000000005		.word 5
0078	0000000000000000	test	
0080	0000000000000000		

Code

Address	Representation	Label	Instruction
0000	60050000		daddi r5,r0,format_
0004	AC050048		sw r5,fs_addr(r0)
0008	60020030		daddi r2,r0,s1
000C	60030038		daddi r3,r0,s2
0010	FC020058		sd r2,s1_addr(r0)
0014	FC030060		sd r3,s2_addr(r0)
0018	600E0048		daddi r14,r0,fs_add
001C	0000014C		syscall 5
0020	0000000C		syscall 0
0024			
0028			
002C			
0030			
0034			
0038			
003C			
0040			

Welcome to EduMIPS64 1.2.7.1 RUNNING

Dr. MIPS

https://brunonova.github.io/drmips/
https://github.com/brunonova/drmips

https://www.youtube.com/watch?v=dyDSV-3v1Ns

2013 1st International Conference of the Portuguese Society for Engineering Education (CISPEE)

Tool to Support Computer Architecture Teaching and Learning

Bruno Nova[1], João C. Ferreira[2], António Araújo[2]
[1]Faculdade de Engenharia da Universidade do Porto, Porto, Portugal
[2]INESC TEC and Faculdade de Engenharia da Universidade do Porto, Porto, Portugal

***Abstract*—Computer architecture is an important subject for informatics and electrical engineering courses. However, students display some difficulties in this subject, mainly due to the lack of educational tools that are intuitive, versatile and graphical. Existing tools are not adequate enough or are very specific. In this paper, an educational MIPS simulator, DrMIPS, is described. This tool simulates the execution of an assembly program on the CPU and displays the datapath graphically. Registers, data memory and assembled code are also displayed and a "performance mode" is also provided. Both unicycle and**

B. Objectives

The main objective of the work presented in this paper was to create a tool to support computer architecture teaching and learning. This educational tool is a simulator the MIPS processor [1], which is a well-known processor in the computer architecture academic community and also one of the most used processors for teaching computer architecture courses in universities [2]. This simulator was developed under the

Dr. MIPS

File View Edit Datapath Execute CPU Help

Code | Assembled | Datapath | Data memory

```
1 li $t0, 10
2 li $t1, 0
3 loop:
4 beq $t1, $t0, end # if t1 == 10 we are done
5 addi $t1, $t1, 1
6 j loop
7 end:
```

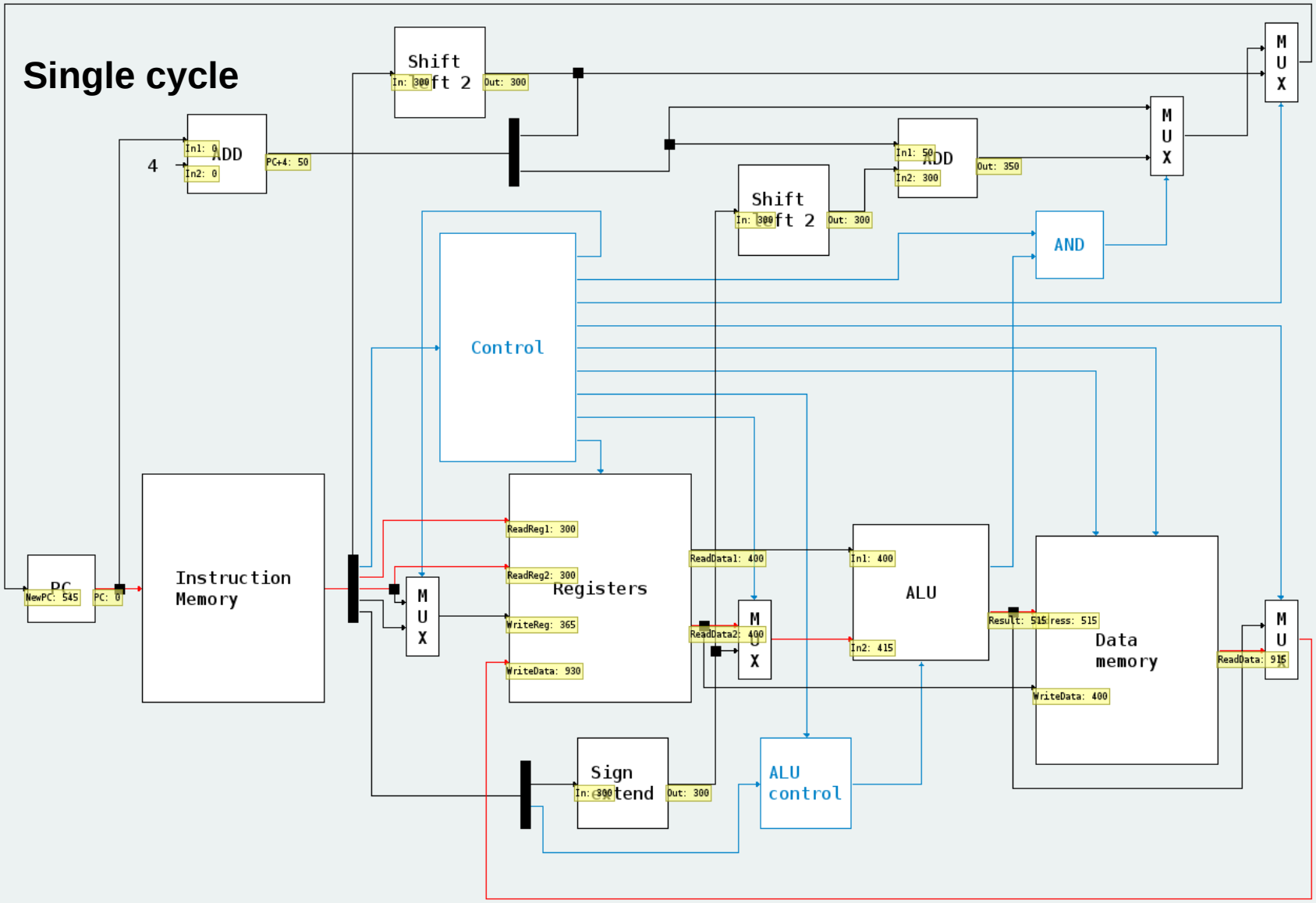
File View Edit Datapath Execute CPU Help

Zoom: 218%

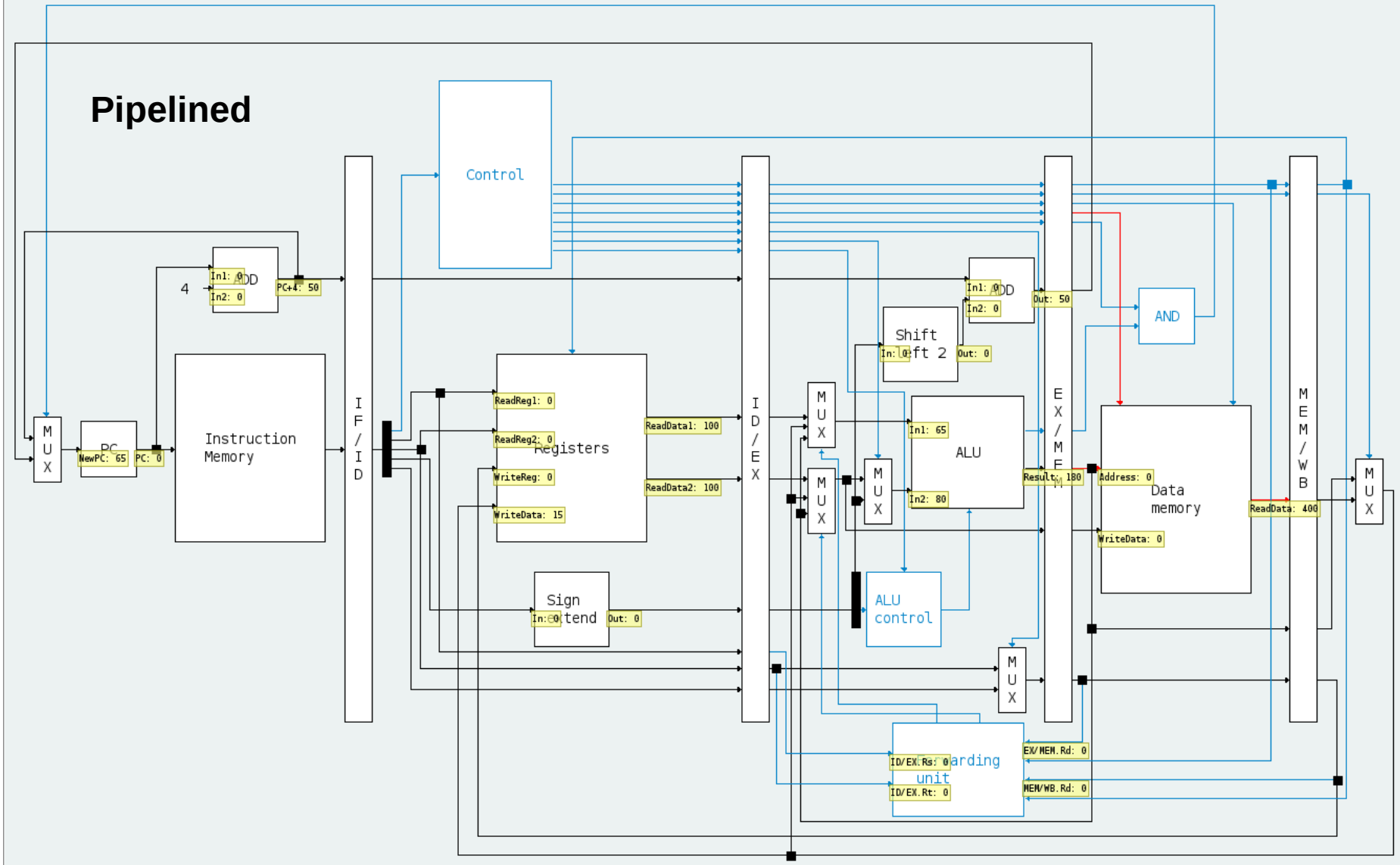
Code | Assembled | Datapath | Data memory

Address	Assembled	Code
0	537395210	1: addi $t0, $0, 10 # li $t0, 10
4	537460736	2: addi $t1, $0, 0 # li $t1, 0
8	287834114	4: loop: beq $t1, $t0, end
12	556335105	5: addi $t1, $t1, 1
16	134217730	6: j loop

Single cycle

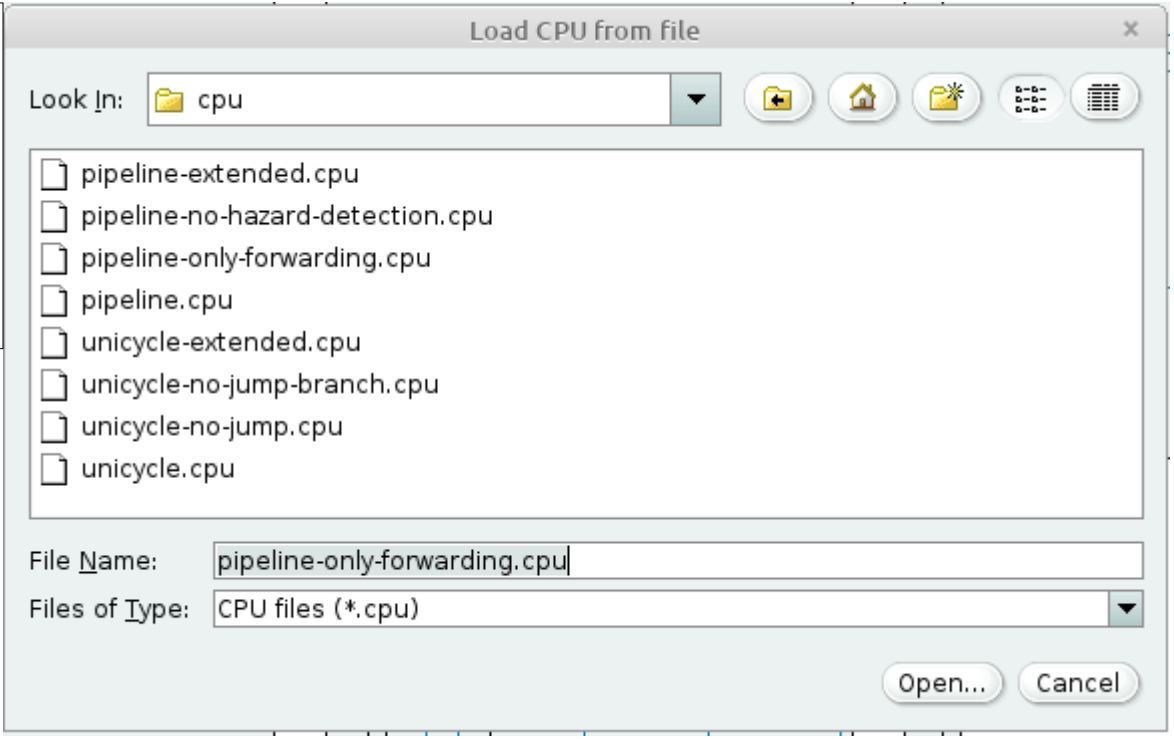


Pipelined


Control
ADD
In1: 0
In2: 0
PC+4: 50
4
MUX
PC
NewPC: 65
PC: 0
Instruction Memory
IF/ID
Registers
ReadReg1: 0
ReadReg2: 0
WriteReg: 0
WriteData: 15
ReadData1: 100
ReadData2: 100
Sign extend
In: 0
Out: 0
ID/EX
MUX
ALU
In1: 65
In2: 80
Result: 180
ALU control
Shift left 2
In: 0
Out: 0
ADD
In1: 0
In2: 0
Out: 50
AND
EX/MEM
Address: 0
Data memory
WriteData: 0
ReadData: 400
MEM/WB
Forwarding unit
ID/EX.Rs: 0
ID/EX.Rt: 0
EX/MEM.Rd: 0
MEM/WB.Rd: 0

Dr. MIPS

There is a support for several micro-architectures
(* but there isn't support for Jump in many of them *)

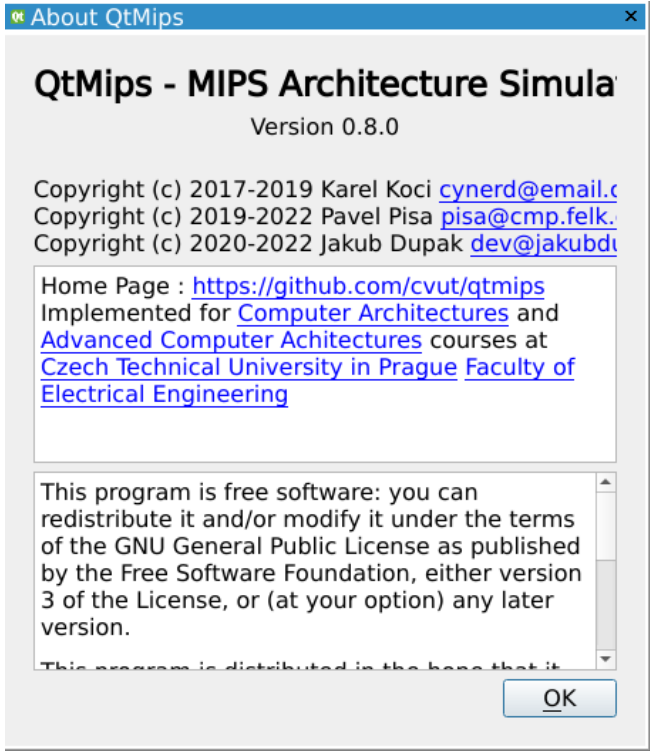


QtMIPS

This is an advanced MIPS simulation tool!
Home page: https://comparch.edu.cvut.cz/qtmips/app/

See a screen capture in the next slide

Tutorial video:
https://www.youtube.com/watch?v=6xH72UBvnaY



QtMIPS

File Machine Windows Help

1x 2x 5x 10x Unlimited Max

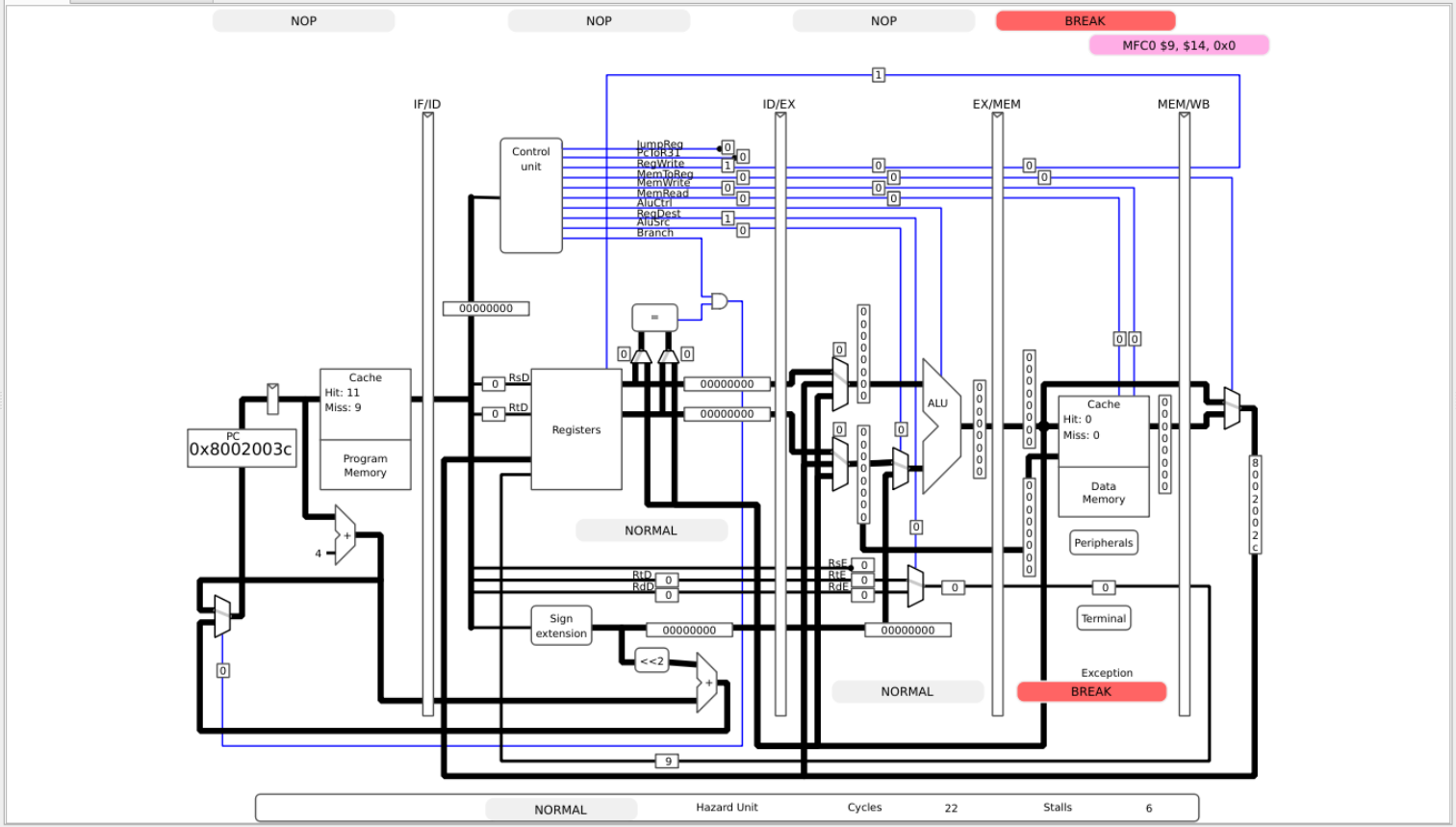
Program

Follow fetch

Bp	Address	Instruction
	0x80020000	ADDI $1, ...
	0x80020004	ADDI $2, ...
	0x80020008	SYNCI ...
	0x8002000C	ADDI $3, ...
	0x80020010	ADDI $4, ...
	0x80020014	LUI $20, ...
	0x80020018	ORI $20, ...
	0x8002001C	MTC0 $20,
	0x80020020	ERET
	0x80020024	ADDI $5, ...
	0x80020028	ADDI $6, ...
	0x8002002C	ADDI $7, ...
	0x80020030	ADDI $8, ...
	0x80020034	MFC0 $9,
	0x80020038	BREAK
	0x8002003C	MFC0 $10,
	0x80020040	LUI $20, ...
	0x80020044	ORI $20, ...
	0x80020048	MTC0 $20,
	0x8002004C	LUI $21, ...
	0x80020050	ORI $21, ...
	0x80020054	LUI $20, ...

0x80020000

Core | cop0-test-ia.S



Memory

Word | Direct

Address	+0
0x00002000	00000000
0x00002004	00000000
0x00002008	00000000
0x0000200C	00000000
0x00002010	00000000
0x00002014	00000000
0x00002018	00000000
0x0000201C	00000000
0x00002020	00000000
0x00002024	00000000
0x00002028	00000000
0x0000202C	00000000
0x00002030	00000000

0x00002000

Terminal

Input:

Part 2: RISC-V

RARS

RARS = RISC-V Assembler and Runtime Simulator

Similar to MARS there is RARS for the RISC-V processor.

Home page: https://github.com/TheThirdOne/rars

Invocation:

java -jar /path/to/the/jar/file/RARS/rars1_4.jar

/home/benjamin/Documents/workspace/rars/examples/bottles.s - RARS 1.1

File Edit Run Settings Tools Help

Run speed at max (no interaction)

Edit Execute

bottles.s printf.s printnum.s printstr.s

```
.globl  main
.data
template:
        .string "%d bottles of beer of the wall. %d bottles of beer. Take one down pass it around. %d bottles of beer on the wall\n"

.text
main:
        # Save state to free up s0, s1, and ra for use
        addi    sp,sp,-16
        sw      s1,0(sp)
        sw      ra,12(sp)

        # Call printf(template, x, x, x-1) for 0 < x < 100
        li      s0,99
loop:
        mv      a1,s0
        addi    s0,s0,-1
        mv      a3,s0
        la      a0,template
        jal     ra,printf
        bnez    s0,loop

        ecall
        ebreak
```

Line: 1 Column: 1 Show Line Numbers

Messages Run I/O

```
99 bottles of beer of the wall. 99 bottles of beer. Take one down pass it around. 98 bottles of beer on the wall
98 bottles of beer of the wall. 98 bottles of beer. Take one down pass it around. 97 bottles of beer on the wall
97 bottles of beer of the wall. 97 bottles of beer. Take one down pass it around. 96 bottles of beer on the wall
96 bottles of beer of the wall. 96 bottles of beer. Take one down pass it around. 95 bottles of beer on the wall
95 bottles of beer of the wall. 95 bottles of beer. Take one down pass it around. 94 bottles of beer on the wall
94 bottles of beer of the wall. 94 bottles of beer. Take one down pass it around. 93 bottles of beer on the wall
93 bottles of beer of the wall. 93 bottles of beer. Take one down pass it around. 92 bottles of beer on the wall
92 bottles of beer of the wall. 92 bottles of beer. Take one down pass it around. 91 bottles of beer on the wall
91 bottles of beer of the wall. 91 bottles of beer. Take one down pass it around. 90 bottles of beer on the wall
90 bottles of beer of the wall. 90 bottles of beer. Take one down pass it around. 89 bottles of beer on the wall
89 bottles of beer of the wall. 89 bottles of beer. Take one down pass it around. 88 bottles of beer on the wall
88 bottles of beer of the wall. 88 bottles of beer. Take one down pass it around. 87 bottles of beer on the wall
87 bottles of beer of the wall. 87 bottles of beer. Take one down pass it around. 86 bottles of beer on the wall
86 bottles of beer of the wall. 86 bottles of beer. Take one down pass it around. 85 bottles of beer on the wall
85 bottles of beer of the wall. 85 bottles of beer. Take one down pass it around. 84 bottles of beer on the wall
84 bottles of beer of the wall. 84 bottles of beer. Take one down pass it around. 83 bottles of beer on the wall
```

Clear

Registers Floating Point Control and Status

Name	Number	Value
zero	0	0x00000000
ra	1	0x00000000
sp	2	0x7fffeffc
gp	3	0x10008000
tp	4	0x00000000
t0	5	0x10010071
t1	6	0x00000000
t2	7	0x00000025
s0	8	0x00000000
s1	9	0x00000000
a0	10	0x00000000
a1	11	0x10010054
a2	12	0x0000001d
a3	13	0x00000000
a4	14	0x00000000
a5	15	0x00000000
a6	16	0x00000000
a7	17	0x0000005d
s2	18	0x00000000
s3	19	0x00000000
s4	20	0x00000000
s5	21	0x00000000
s6	22	0x00000000
s7	23	0x00000000
s8	24	0x00000000
s9	25	0x00000000
s10	26	0x00000000
s11	27	0x00000000
t3	28	0x00000030
t4	29	0x00000000
t5	30	0x00000000
t6	31	0x00000000
pc	32	0x00400054

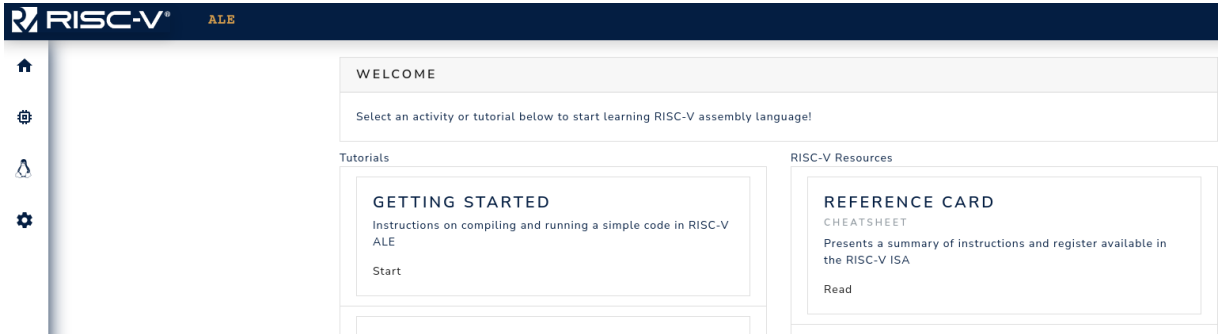
RISC-V

Home page: https://riscv-programming.org/ale/#home
(it was tested on Google Chrome)

This is a portal for executing RISC-V code.

<u>RISC-V emulation is done here in two stages:</u>
1) build: This is done on the local (Linux) machine with the RISC-V tool chain.
2) Upload and execute the code in the portal.

<u>A video tutorial:</u>
https://www.youtube.com/watch?v=Av5kg5xavuo



RISC-V

QtRVSim

QtRVSim is the counter part to the QtMIPS simulator

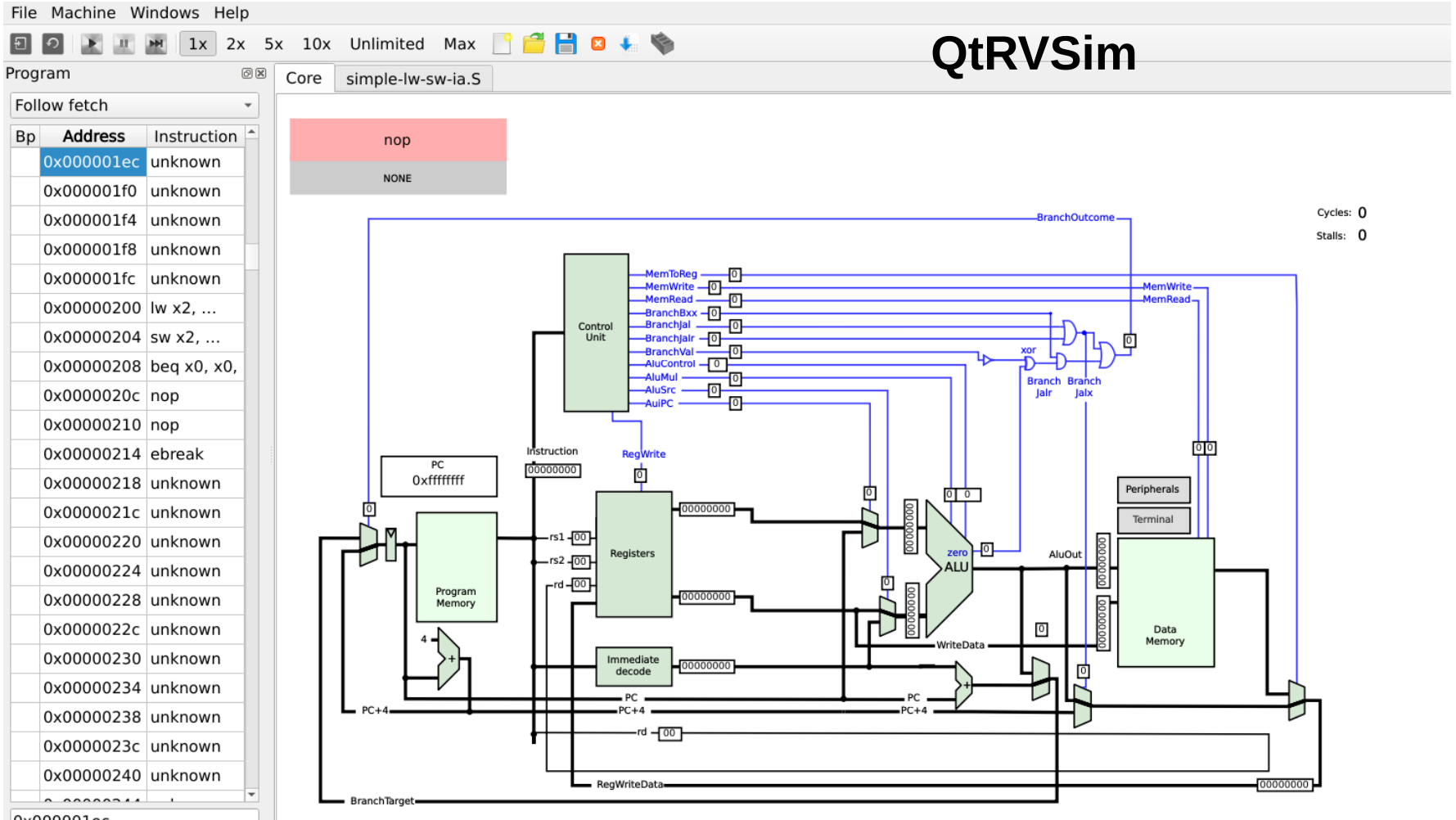
Tutorial are available at:
https://comparch.edu.cvut.cz/slides/ewc22-qtrvsim.pdf
and:
https://drive.google.com/file/d/1Rxd6-Qm6LhkDjd9740IAkY_kX1roLOeO/view

The simulator is available at:
https://github.com/cvut/qtrvsim
and must be downloaded to the local computer.

In addition there is an online version (like in the case of the QtMIPS) is available from here:
https://comparch.edu.cvut.cz/qtrvsim/app/

RISC-V

QtRVSim



Ripes

A graphical processor simulator and assembly editor for the RISC-V ISA

Homepage:
https://github.com/mortbopet/Ripes/

Documentation:
https://github.com/mortbopet/Ripes/tree/master/docs

Easy install:
In Linux as AppImage just do
chmod u+x Ripes-v2.2.6-linux-x86_64.AppImage

File Edit View Help

100 ms

Editor

Processor

Cache

Memory

I/O

Input type: Assembly C

```
1  # This example shows an implementation of the mathematical
2  # factorial function (! function) to find the factorial value of !7 = 5040.
3
4  .data
5  argument: .word   7
6  str1:     .string "Factorial value of "
7  str2:     .string " is "
8
9  .text
10 main:
11         lw  a0, argument   # Load argument from static data
12         jal ra, fact       # Jump-and-link to the 'fact' label
13
14         # Print the result to console
15         mv  a1, a0
16         lw  a0, argument
17         jal ra, printResult
18
19         # Exit program
20         li a7, 10
21         ecall
22
23 fact:
24         addi sp, sp, -16
25         sw   ra, 8(sp)
26         sw   a0, 0(sp)
27         addi t0, a0, -1
28         bge  t0, zero, nfact
29
30         addi a0, zero, 1
31         addi sp, sp, 16
32         jr x1
33
34 nfact:
35         addi a0, a0, -1                                    EX
36         jal  ra, fact                                      ID
37         addi t1, a0, 0                                     IF
```

Executable code

View mode: Binary Disassembled

```
    0:        10000517        auipc x10 0x10000
    4:        00052503        lw x10 0 x10
    8:        01c000ef        jal x1 28 <fact>
    c:        00050593        addi x11 x10 0
    10:        10000517        auipc x10 0x10000
    14:        ff052503        lw x10 -16 x10
    18:        04c000ef        jal x1 76 <printResult>
    1c:        00a00893        addi x17 x0 10
    20:        00000073        ecall

00000024 <fact>:
    24:        ff010113        addi x2 x2 -16
    28:        00112423        sw x1 8 x2
    2c:        00a12023        sw x10 0 x2
    30:        fff50293        addi x5 x10 -1
    34:        0002d863        bge x5 x0 16 <nfact>
    38:        00100513        addi x10 x0 1
    3c:        01010113        addi x2 x2 16
    40:        00008067        jalr x0 x1 0

00000044 <nfact>:
    44:        fff50513        addi x10 x10 -1                    EX
    48:        fddff0ef        jal x1 -36 <fact>                  ID
    4c:        00050333        addi x6 x10 0                      IF
    50:        00012503        lw x10 0 x2
    54:        00812083        lw x1 8 x2
    58:        01010113        addi x2 x2 16
    5c:        02650533        mul x10 x10 x6
    60:        00008067        jalr x0 x1 0

00000064 <printResult>:
    64:        00050293        addi x5 x10 0
    68:        00058313        addi x6 x11 0
    6c:        10000517        auipc x10 0x10000
    70:        f9850513        addi x10 x10 -104
    74:        00400893        addi x17 x0 4
    78:        00000073        ecall
```

GPR

Name	Alias	Value
x0	zero	0x00000000
x1	ra	0x0000004c
x2	sp	0x7fffff90
x3	gp	0x10000000
x4	tp	0x00000000
x5	t0	0x00000001
x6	t1	0x00000000
x7	t2	0x00000000
x8	s0	0x00000000
x9	s1	0x00000000
x10	a0	0x00000002
x11	a1	0x00000000
x12	a2	0x00000000
x13	a3	0x00000000
x14	a4	0x00000000
x15	a5	0x00000000
x16	a6	0x00000000
x17	a7	0x00000000
x18	s2	0x00000000
x19	s3	0x00000000
x20	s4	0x00000000
x21	s5	0x00000000
x22	s6	0x00000000
x23	s7	0x00000000
x24	s8	0x00000000
x25	s9	0x00000000

Hex

Console

```
Factorial value of 7 is 5040
Program exited with code: 0
```

The assembly code

The disassembled code

The registers

ISA: RV32IM

File Edit View Help

100 ms

Editor
Processor
Cache
Memory
I/O

5-Stage RISC-V Processor

addi x6 x10 0 | jal x1 -36 <fact> | addi x10 x10 -1 | nop (flush) | nop (flush)

PC
Instr. memory
Compressed decoder
IFID
Decode
Registers
Imm.
IDEX
ALU
Branch
EX/MEM
Data memory
MEM/WB

Registers

GPR

Name	Alias	Value
x0	zero	0x00000000
x1	ra	0x0000004c
x2	sp	0x7fffff90
x3	gp	0x10000000
x4	tp	0x00000000
x5	t0	0x00000001
x6	t1	0x00000000
x7	t2	0x00000000
x8	s0	0x00000000
x9	s1	0x00000000
x10	a0	0x00000002
x11	a1	0x00000000
x12	a2	0x00000000
x13	a3	0x00000000
x14	a4	0x00000000
x15	a5	0x00000000
x16	a6	0x00000000
x17	a7	0x00000000
x18	s2	0x00000000

Display type: Hex

Console

```
Factorial value of 7 is 5040
Program exited with code: 0
```

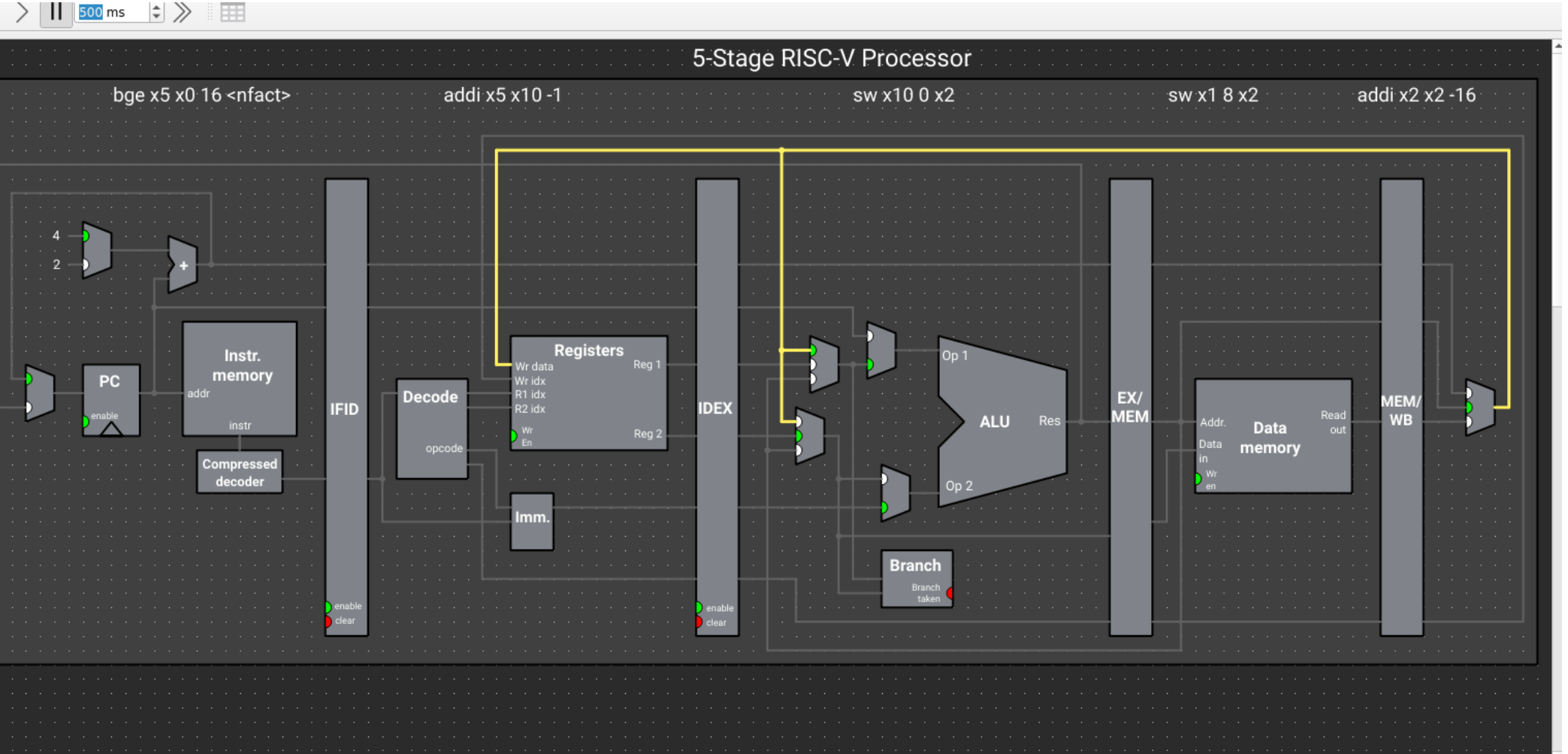
Execution info

Cycles:	69
Instrs. retired:	43
CPI:	1.6
IPC:	0.623
Clock rate:	1.05 Hz

Instruction memory

BP	Addr	Stage	Instruction
	0x3c		addi x2 x2 16
	0x40		jalr x0 x1 0
	0x44	EX	addi x10 x10 -1
	0x48	ID	jal x1 -36 <fact>
	0x4c	IF	addi x6 x10 0
	0x50		lw x10 0 x2

Processor: 5-stage processor ISA: RV32IM


500 ms
5-Stage RISC-V Processor
bge x5 x0 16 <nfact>
addi x5 x10 -1
sw x10 0 x2
sw x1 8 x2
addi x2 x2 -16
4
2
+
PC
enable
Instr.
memory
addr
instr
Compressed
decoder
IFID
enable
clear
Decode
opcode
Registers
Wr data
Wr idx
R1 idx
R2 idx
Wr
En
Reg 1
Reg 2
Imm.
IDEX
enable
clear
Op 1
ALU
Res
Op 2
Branch
Branch
taken
EX/
MEM
Addr.
Data
in
Wr
en
Data
memory
Read
out
MEM/
WB

Part 3: Additional simulators

Reorder Buffer

Tomasulo Algorithm

Reorder Buffer

https://www.ecs.umass.edu/ece/koren/architecture/ROB/rob_simulator.htm

האתר הזה חצי חי

Simulation Control

+1 Clock Cycle

+5 Clock Cycles

current clock:

Simulation Settings

START SIMULATION

CONFIGURATION

HELP

Reorder Buffer Simulator

Configuration Window

Number of **ROB** entries: 6

FUs	Number	Ex. Cycles	# of **RS**s
FP-Adder	1	2	3
FP-Multiplier	1	10	1
FP-Divider	1	40	1

Number of **CDB**s: 1

Number of Load Buffers: 2

Latency of Load: 2 clock cycle(s)

change # of Instructions to 6

current # of Instructions: 6

Examples

Instructions

#	Opcode	Operands			causes an exception
		Dest.	Op1	Op2	
1	L.D	F6	F0	F0	☐
2	L.D	F2	F0	F0	☐
3	MUL.D	F0	F2	F4	☐
4	SUB.D	F8	F6	F2	☐
5	DIV.D	F10	F0	F6	☐
6	ADD.D	F6	F8	F2	☐

Tomasulo Algorithm

http://nathantypanski.github.io/tomasulo-simulator/

Tomasulo algorithm simulator (protoype)

This simulates Tomasulo's algorithm for a floating-point MIPS-like instruction pipeline, demonstrating out-of-order execution. The source is on GitHub.

Click instructions on the right to issue and execute them. **Instructions will only execute if all of their data dependencies have been resolved**, but they may issue in any order (though at least issuing them in order is recommended). Currently, loads have two-step execution and still require a writeback cycle. Regs[x] is the value at location x from the register file.

Color codes are as follows: *destination* *source* *occupied source* *occupied destination*.

Reservation stations

Name	Busy	Op	Vj	Vk	Qj	Qk	A	Result
Load0	false							
Load1	false							
Add0	false							
Add1	false							
Add2	false							
Mult0	false							
Mult1	false							
Store0	false							
Store1	false							

Instruction Status

Instruction	Issue	Execute	Write result
L.D F6,32(R2)			
L.D F2,44(R3)			
MUL.D F0,F2,F4			
ADD.D F10,F12,F8			
SUB.D F8,F2,F6			
DIV.D F10,F0,F6			
ADD.D F6,F8,F2			
S.D F6,32(R2)			

Tomasulo Algorithm

https://www.ecs.umass.edu/ece/koren/architecture/Tomasulo/AppletTomasulo.html

Can work from Chrome but requires Java Plug-in.

For example: Cheerp Java Applet Runner.

Dynamic Scheduling Using Tomasulo's Algorithm