San Jose State University
Department of Computer Engineering

CMPE 140 Lab Report

Lab 7 Report

Title Enhanced Single-cycle MIPS Processor

Semester Fall **Date** 03/24/2020

by

Name Thien N Hoang **SID** 012555673
(typed) *(typed)*

Name Phat Le **SID** 012067666
(typed) *(typed)*

Lab Checkup Record

Week	Performed By (signature)	Checked By (signature)	Tasks Successfully Completed*	Tasks Partially Completed*	Tasks Failed or Not Performed*
1	[illegible] TH				
2					

*** Detailed descriptions must be given in the report.**

Authors

	Thien Hoang I was born and raised in Ho Chi Minh City, Viet Name. I came to the United States when I was 16 years old. Before transferring to SJSU, I lived in Houston Texas with my family. I am currently a 5th-year Computer Engineering major at San Jose State University. I have an interest in building the circuit and aspire to be a circuit designer in the future. I enjoy riding a motorcycle on the weekend. It helps me release my stress
	Phat Le I am an international student from Vietnam, also a transfer student from Seattle, and currently a senior Computer Engineering student at San Jose State University. My primary goal is to invent a device that will have huge improvements in humanity. By understanding the way computer software and hardware works, I believe my dream can become true one day. I also enjoy playing RPG console video games and singing alone with my guitar. Sounds very introvert right?!

I. Introduction

The purpose of this lab is to extend the initial design of the single-cycle MIPS processor from previous labs. Specifically, the requirements are to modify the initial design of block diagrams, truth table as well as performing functional verification and hardware validation of the new design. This report will cover the first part of the lab which is modifying the block diagram and the truth table.

II. Design methodology

The first task of this lab is to extend the architecture of the given single-cycle MIPS microprocessor to support more MIPS instruction. Beside the original instruction(`add, sub, and, or, slt, lw, sw, beq, j, addi`), the new design would cover the following instruction (`MULTU, MFHI, MFLO, JR, JAL, SLL, SLR`). Specifically, the task is to modify the microprocessor blocks diagram as well as modify the truth table. The additional information and modifications would have different colors than the initial black diagram as shown in *Figures 1* and *2* below.

According to the MIPS Reference Datasheet, `MULTU` instruction should read values from registers Rs and Rt, and then store the multiplied result to `hi` register and `lo` register. Therefore, a new register module(hilo_reg) was added to represent `hi` and `lo` registers. Since the alu have not supported multiplication operation as well as outputting a 64-bit result, the design was added a multiplication module(multu) to take in two 32-bit inputs and one 64-bit output. `MFHI` and `MFLO` then can access the upper or lower bit from hilo_reg output using a multiplexer (multu_mux). In addition, the multu_mux also selected the initial write to register file signal (wd_rf) so the result could be written to the register file.

For the `JR` instruction, the intended jump counter address is the data stored in Rs(alu_pa). Therefore, it should directly connect to the program counter register(pc_reg). For this reason, another 2-to-1 mux was necessary to select between the alu_pa and the initial program counter (pc).

The main functionality of `JAL` instruction is to jump to the desired destination (same as `j` instruction) and store the next instruction's address to $ra. As a result, the extended diagram should make the register file(rf) to select the 31th ($ra) as the write register address, and also select the pc+4 as the write data.

Lastly, for the `SLL` and `SLR` instruction, the value of Rt is shifted based on the shamt portion(instr[10:6]) in which the result would be stored at Rd. For this reason, the alu was extended to support logical shift left and right operation. Another 2-to-1 mux(sl_mux) was also required so that the alu inputs would be the value stored at Rt and the shamt value.

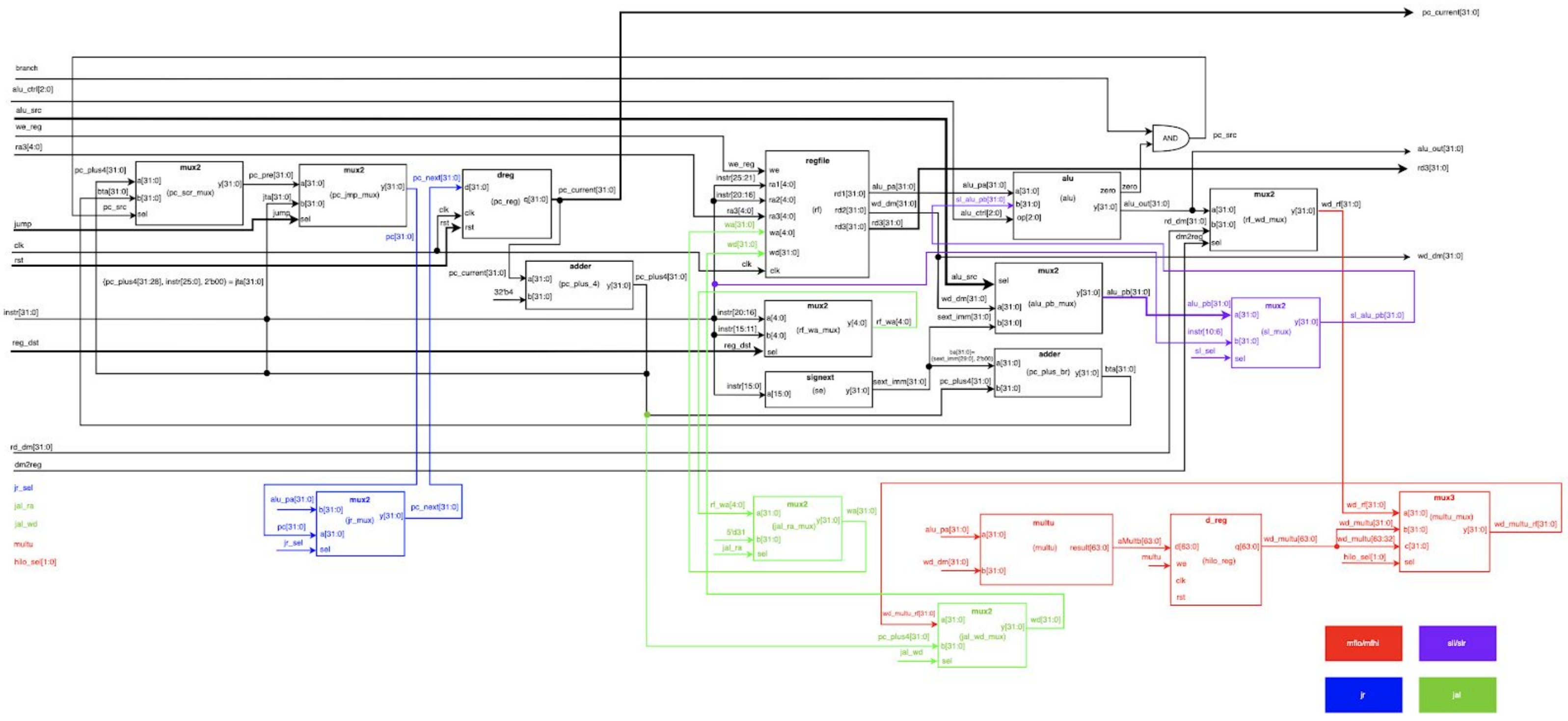


Figure 1: Extended MIPS microarchitecture

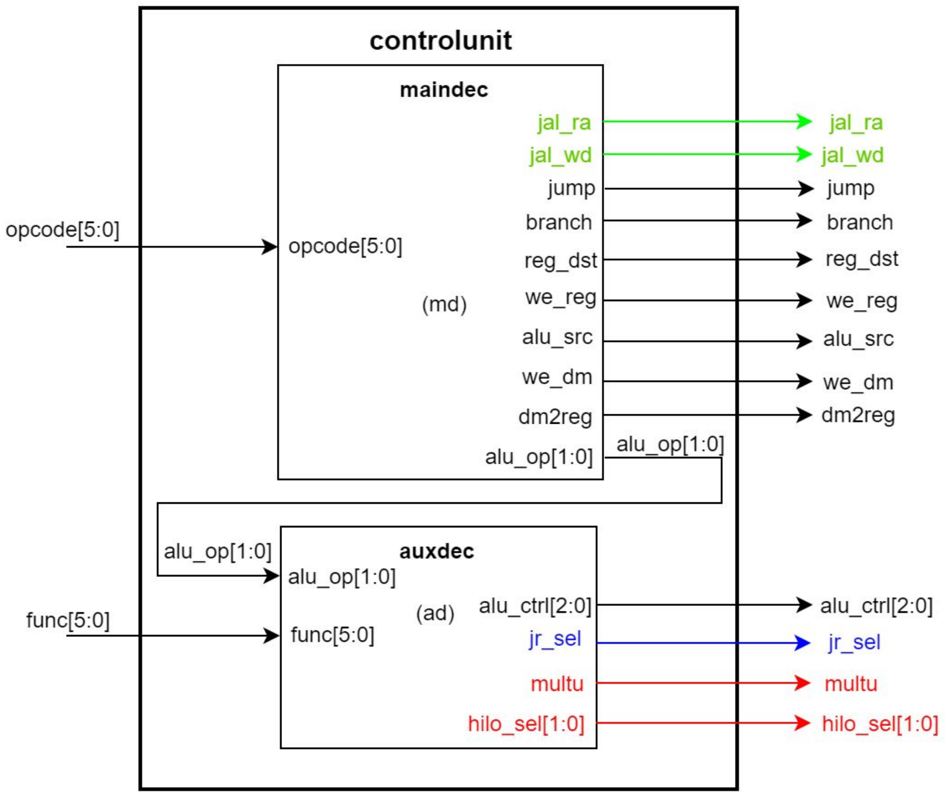


Figure 2:Extended MIPS microarchitecture - Control Unit

For the control unit truth table, the decision of putting the instruction into the main decoder table or into the auxiliary decoder table depends on the type of the instruction as well as its opcode and its function portion as shown in *Table 1* and *2* below. Moreover, most of the new signals are for the mux select signal which would become 1 when the related instruction was performing. Since the alu logical shift left and right was added, more alu control signals were also added (sll = 011, srl = 100).

Table 1: Main Decoder Truth Table

maindec											
Instruction	**Opcode**	**we_reg**	**reg_dst**	**alu_src**	**branch**	**we_dm**	**dm2reg**	**alu_op [1:0]**	**Jump**	**jal_ra**	**jal_wd**
j	00_0010	0	0	0	0	0	0	00	1	0	0
jal	00_0011	1	0	0	0	0	0	00	1	1	1
beq	00_0100	0	0	0	1	0	0	01	0	0	0
addi	00_1000	1	0	1	0	0	0	00	0	0	0
R-type	00_0000	1	1	0	0	0	0	10	0	0	0
lw	10_0011	1	0	1	0	0	1	00	0	0	0
sw	10_1011	0	0	1	0	1	0	00	0	0	0

Table 2:Auxiliary Decoder Truth Table

auxdec						
Instruction	**funct**	**alu_ctrl[2:0]**	**multu**	**hilo_sel[1:0]**	**sl_sel**	**jr_sel**
mfhi	00_1010	xxx	0	11	0	0
mflo	00_1100	xxx	0	10	0	0
sll	00_0000	011	0	00	1	0
srl	00_0011	100	0	00	1	0
jr	00_0100	xxx	0	00	0	1
multu	01_0011	xxx	1	00	0	0
and	10_0100	000	0	00	0	0
or	10_0101	001	0	00	0	0
add	10_0000	010	0	00	0	0
sub	10_0010	110	0	00	0	0
slt	10_0101	111	0	00	0	0

III. Conclusion

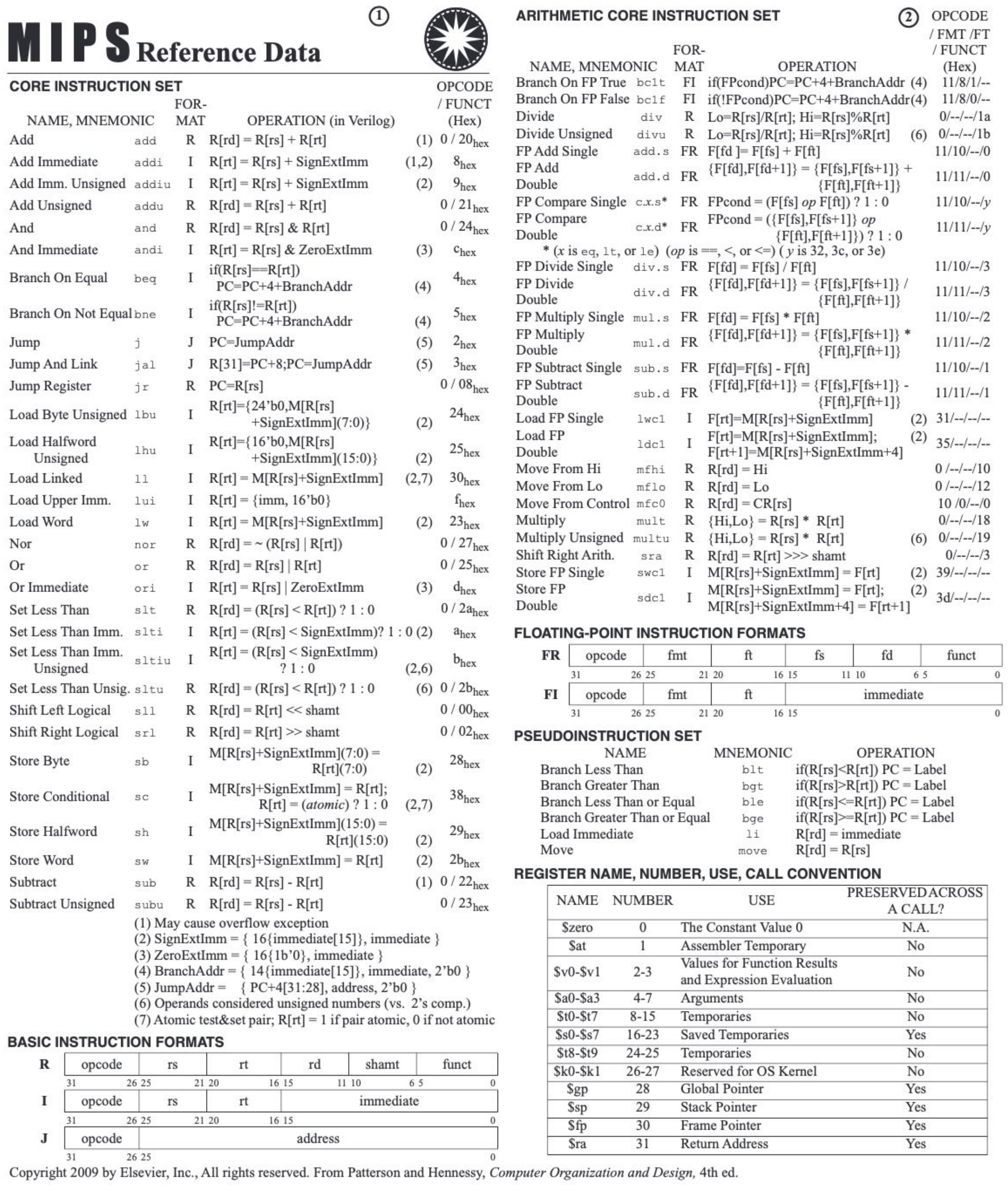
In sum, this lab helps us understand the microarchitecture designing process better by just observing the behavior of the datapath and the MIPS reference datasheet. We were able to learn the difference between the main decoder and auxiliary decoder by adding more instructions to them. In addition, we were able to learn the difference between the single-cycle and pipelining MIPS processor as we were dealing with `JAL` instruction. While the single-cycle system uses PC+4 to be stored in $ra, the pipelining system requires PC+8 to be stored in $ra since it skips one more clock cycle for stalling. Lastly, our design could have been better if we were able to use a single mux(monolithic) to control all the new signals instead of multiple muxes(ladder) since each of the muxes has a propagation delay period.

IV. Successful Task

1. We were able to do the official draft of extended MIPS microarchitecture
2. We were able to create a control unit truth table

V. Appendix

All of the modifications of the MIPS instruction operation were based on the MIPS Reference Data shown below.



① **MIPS Reference Data**

CORE INSTRUCTION SET

NAME, MNEMONIC		FOR-MAT	OPERATION (in Verilog)		OPCODE / FUNCT (Hex)
Add	add	R	R[rd] = R[rs] + R[rt]	(1)	0 / 20_{hex}
Add Immediate	addi	I	R[rt] = R[rs] + SignExtImm	(1,2)	8_{hex}
Add Imm. Unsigned	addiu	I	R[rt] = R[rs] + SignExtImm	(2)	9_{hex}
Add Unsigned	addu	R	R[rd] = R[rs] + R[rt]		0 / 21_{hex}
And	and	R	R[rd] = R[rs] & R[rt]		0 / 24_{hex}
And Immediate	andi	I	R[rt] = R[rs] & ZeroExtImm	(3)	c_{hex}
Branch On Equal	beq	I	if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(4)	4_{hex}
Branch On Not Equal	bne	I	if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	(4)	5_{hex}
Jump	j	J	PC=JumpAddr	(5)	2_{hex}
Jump And Link	jal	J	R[31]=PC+8;PC=JumpAddr	(5)	3_{hex}
Jump Register	jr	R	PC=R[rs]		0 / 08_{hex}
Load Byte Unsigned	lbu	I	R[rt]={24'b0,M[R[rs] +SignExtImm](7:0)}	(2)	24_{hex}
Load Halfword Unsigned	lhu	I	R[rt]={16'b0,M[R[rs] +SignExtImm](15:0)}	(2)	25_{hex}
Load Linked	ll	I	R[rt] = M[R[rs]+SignExtImm]	(2,7)	30_{hex}
Load Upper Imm.	lui	I	R[rt] = {imm, 16'b0}		f_{hex}
Load Word	lw	I	R[rt] = M[R[rs]+SignExtImm]	(2)	23_{hex}
Nor	nor	R	R[rd] = ~ (R[rs] | R[rt])		0 / 27_{hex}
Or	or	R	R[rd] = R[rs] | R[rt]		0 / 25_{hex}
Or Immediate	ori	I	R[rt] = R[rs] | ZeroExtImm	(3)	d_{hex}
Set Less Than	slt	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0		0 / $2a_{hex}$
Set Less Than Imm.	slti	I	R[rt] = (R[rs] < SignExtImm)? 1 : 0	(2)	a_{hex}
Set Less Than Imm. Unsigned	sltiu	I	R[rt] = (R[rs] < SignExtImm) ? 1 : 0	(2,6)	b_{hex}
Set Less Than Unsig.	sltu	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0	(6)	0 / $2b_{hex}$
Shift Left Logical	sll	R	R[rd] = R[rt] << shamt		0 / 00_{hex}
Shift Right Logical	srl	R	R[rd] = R[rt] >> shamt		0 / 02_{hex}
Store Byte	sb	I	M[R[rs]+SignExtImm](7:0) = R[rt](7:0)	(2)	28_{hex}
Store Conditional	sc	I	M[R[rs]+SignExtImm] = R[rt]; R[rt] = (*atomic*) ? 1 : 0	(2,7)	38_{hex}
Store Halfword	sh	I	M[R[rs]+SignExtImm](15:0) = R[rt](15:0)	(2)	29_{hex}
Store Word	sw	I	M[R[rs]+SignExtImm] = R[rt]	(2)	$2b_{hex}$
Subtract	sub	R	R[rd] = R[rs] - R[rt]	(1)	0 / 22_{hex}
Subtract Unsigned	subu	R	R[rd] = R[rs] - R[rt]		0 / 23_{hex}

(1) May cause overflow exception
(2) SignExtImm = { 16{immediate[15]}, immediate }
(3) ZeroExtImm = { 16{1b'0}, immediate }
(4) BranchAddr = { 14{immediate[15]}, immediate, 2'b0 }
(5) JumpAddr = { PC+4[31:28], address, 2'b0 }
(6) Operands considered unsigned numbers (vs. 2's comp.)
(7) Atomic test&set pair; R[rt] = 1 if pair atomic, 0 if not atomic

BASIC INSTRUCTION FORMATS

R	opcode	rs	rt	rd	shamt	funct
	31 26	25 21	20 16	15 11	10 6	5 0
I	opcode	rs	rt	immediate		
	31 26	25 21	20 16	15 0		
J	opcode	address				
	31 26	25 0				

ARITHMETIC CORE INSTRUCTION SET ②

NAME, MNEMONIC		FOR-MAT	OPERATION		OPCODE / FMT /FT / FUNCT (Hex)
Branch On FP True	bc1t	FI	if(FPcond)PC=PC+4+BranchAddr	(4)	11/8/1/--
Branch On FP False	bc1f	FI	if(!FPcond)PC=PC+4+BranchAddr	(4)	11/8/0/--
Divide	div	R	Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt]		0/--/--/1a
Divide Unsigned	divu	R	Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt]	(6)	0/--/--/1b
FP Add Single	add.s	FR	F[fd]= F[fs] + F[ft]		11/10/--/0
FP Add Double	add.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} + {F[ft],F[ft+1]}		11/11/--/0
FP Compare Single	c.*x*.s*	FR	FPcond = (F[fs] *op* F[ft]) ? 1 : 0		11/10/--/*y*
FP Compare Double	c.*x*.d*	FR	FPcond = ({F[fs],F[fs+1]} *op* {F[ft],F[ft+1]}) ? 1 : 0		11/11/--/*y*
* (*x* is eq, lt, or le) (*op* is ==, <, or <=) (*y* is 32, 3c, or 3e)					
FP Divide Single	div.s	FR	F[fd] = F[fs] / F[ft]		11/10/--/3
FP Divide Double	div.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} / {F[ft],F[ft+1]}		11/11/--/3
FP Multiply Single	mul.s	FR	F[fd] = F[fs] * F[ft]		11/10/--/2
FP Multiply Double	mul.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} * {F[ft],F[ft+1]}		11/11/--/2
FP Subtract Single	sub.s	FR	F[fd]=F[fs] - F[ft]		11/10/--/1
FP Subtract Double	sub.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} - {F[ft],F[ft+1]}		11/11/--/1
Load FP Single	lwc1	I	F[rt]=M[R[rs]+SignExtImm]	(2)	31/--/--/--
Load FP Double	ldc1	I	F[rt]=M[R[rs]+SignExtImm]; F[rt+1]=M[R[rs]+SignExtImm+4]	(2)	35/--/--/--
Move From Hi	mfhi	R	R[rd] = Hi		0 /--/--/10
Move From Lo	mflo	R	R[rd] = Lo		0 /--/--/12
Move From Control	mfc0	R	R[rd] = CR[rs]		10 /0/--/0
Multiply	mult	R	{Hi,Lo} = R[rs] * R[rt]		0/--/--/18
Multiply Unsigned	multu	R	{Hi,Lo} = R[rs] * R[rt]	(6)	0/--/--/19
Shift Right Arith.	sra	R	R[rd] = R[rt] >>> shamt		0/--/--/3
Store FP Single	swc1	I	M[R[rs]+SignExtImm] = F[rt]	(2)	39/--/--/--
Store FP Double	sdc1	I	M[R[rs]+SignExtImm] = F[rt]; M[R[rs]+SignExtImm+4] = F[rt+1]	(2)	3d/--/--/--

FLOATING-POINT INSTRUCTION FORMATS

FR	opcode	fmt	ft	fs	fd	funct
	31 26	25 21	20 16	15 11	10 6	5 0
FI	opcode	fmt	ft	immediate		
	31 26	25 21	20 16	15 0		

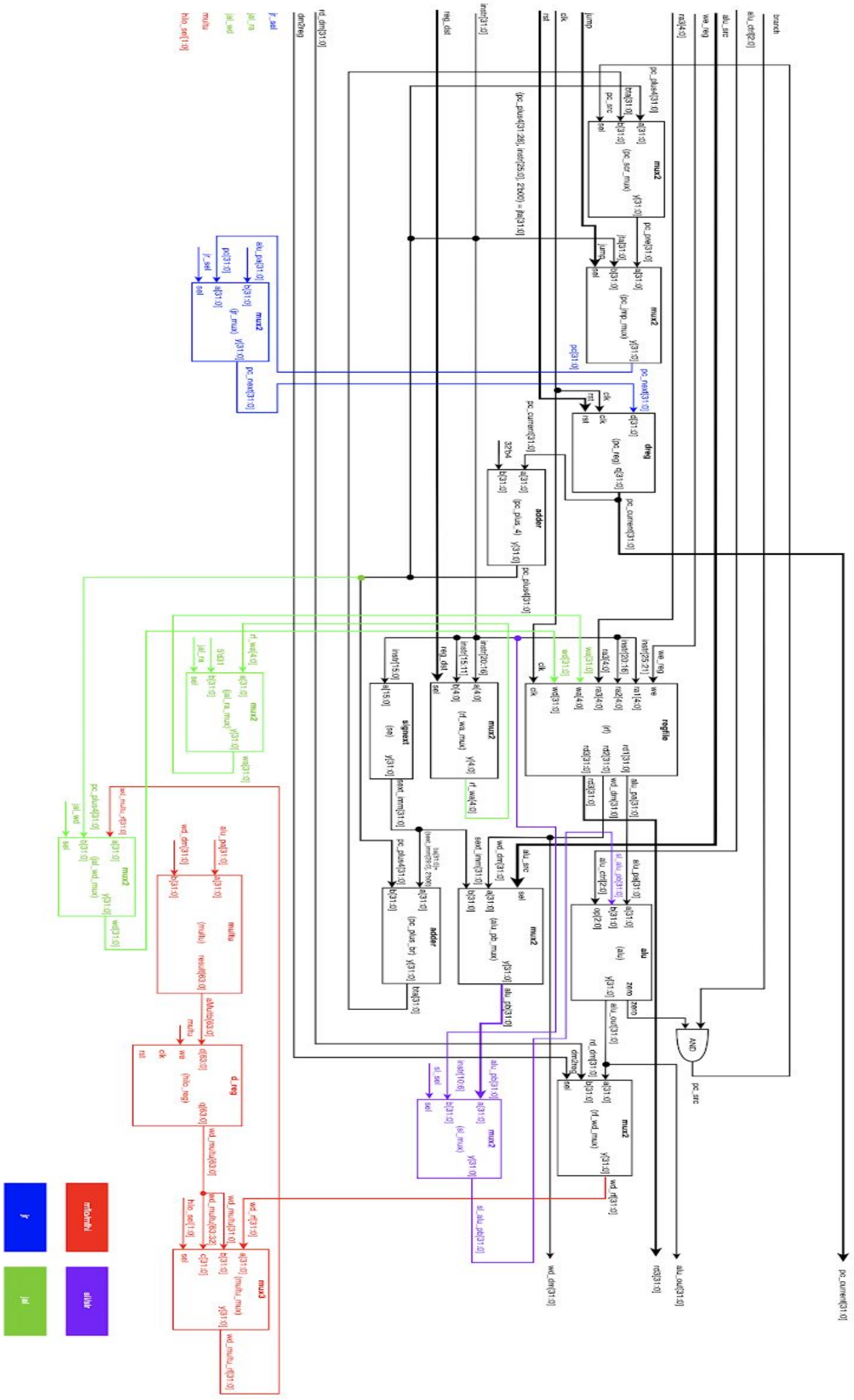
PSEUDOINSTRUCTION SET

NAME	MNEMONIC	OPERATION
Branch Less Than	blt	if(R[rs]<R[rt]) PC = Label
Branch Greater Than	bgt	if(R[rs]>R[rt]) PC = Label
Branch Less Than or Equal	ble	if(R[rs]<=R[rt]) PC = Label
Branch Greater Than or Equal	bge	if(R[rs]>=R[rt]) PC = Label
Load Immediate	li	R[rd] = immediate
Move	move	R[rd] = R[rs]

REGISTER NAME, NUMBER, USE, CALL CONVENTION

NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
$zero	0	The Constant Value 0	N.A.
$at	1	Assembler Temporary	No
$v0-$v1	2-3	Values for Function Results and Expression Evaluation	No
$a0-$a3	4-7	Arguments	No
$t0-$t7	8-15	Temporaries	No
$s0-$s7	16-23	Saved Temporaries	Yes
$t8-$t9	24-25	Temporaries	No
$k0-$k1	26-27	Reserved for OS Kernel	No
$gp	28	Global Pointer	Yes
$sp	29	Stack Pointer	Yes
$fp	30	Frame Pointer	Yes
$ra	31	Return Address	Yes

The bigger datapath block diagram is also included below for better observation.


mux2
dreg
adder
regfile
signext
alu
AND
mux3
multu
d_reg
mflo/mfhi
sll/slr
jr
jal