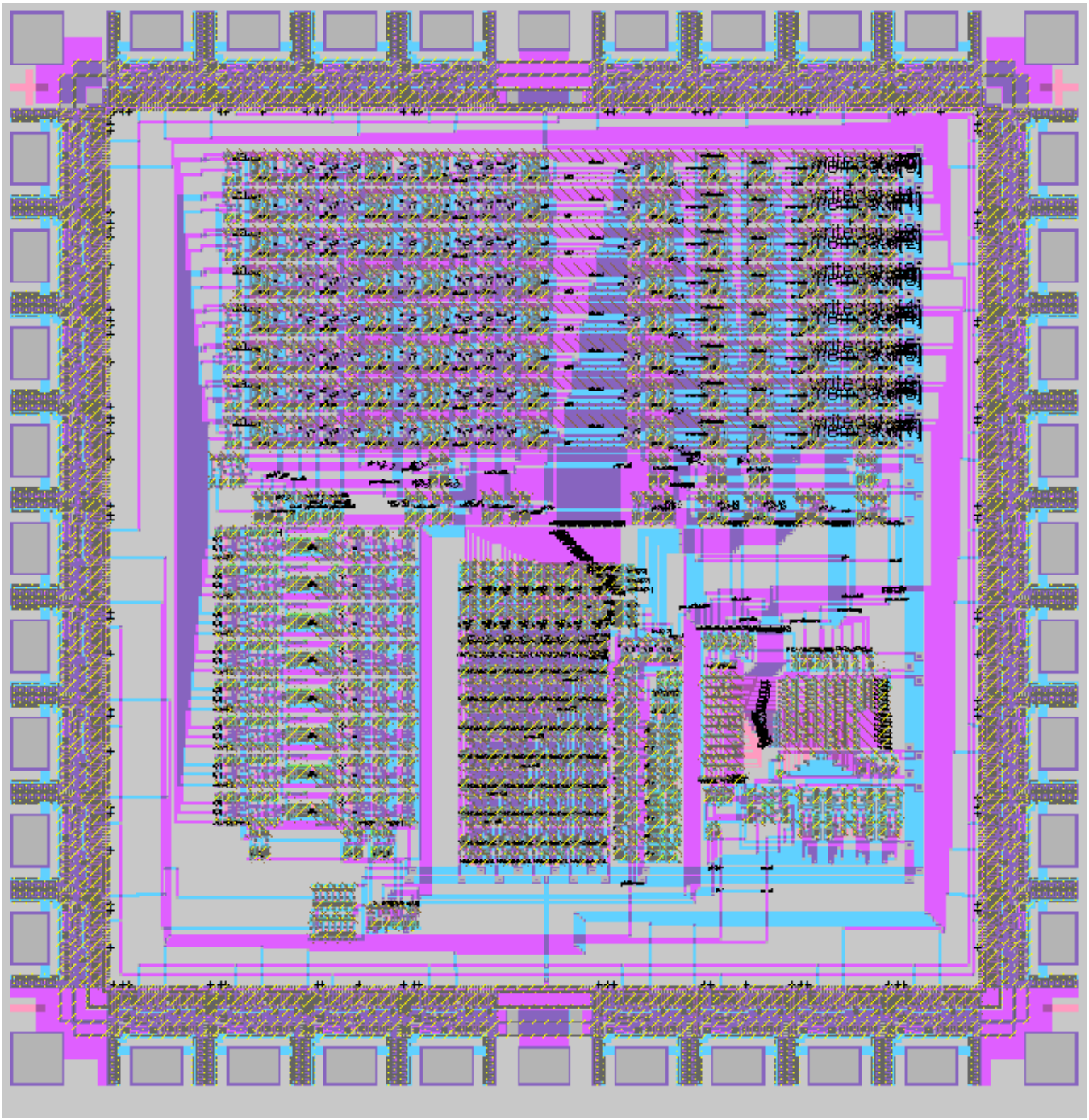
Mini MIPS Microprocessor

Daniel Lee
Sean Kao
VLSI Spring 2001



Functional Overview

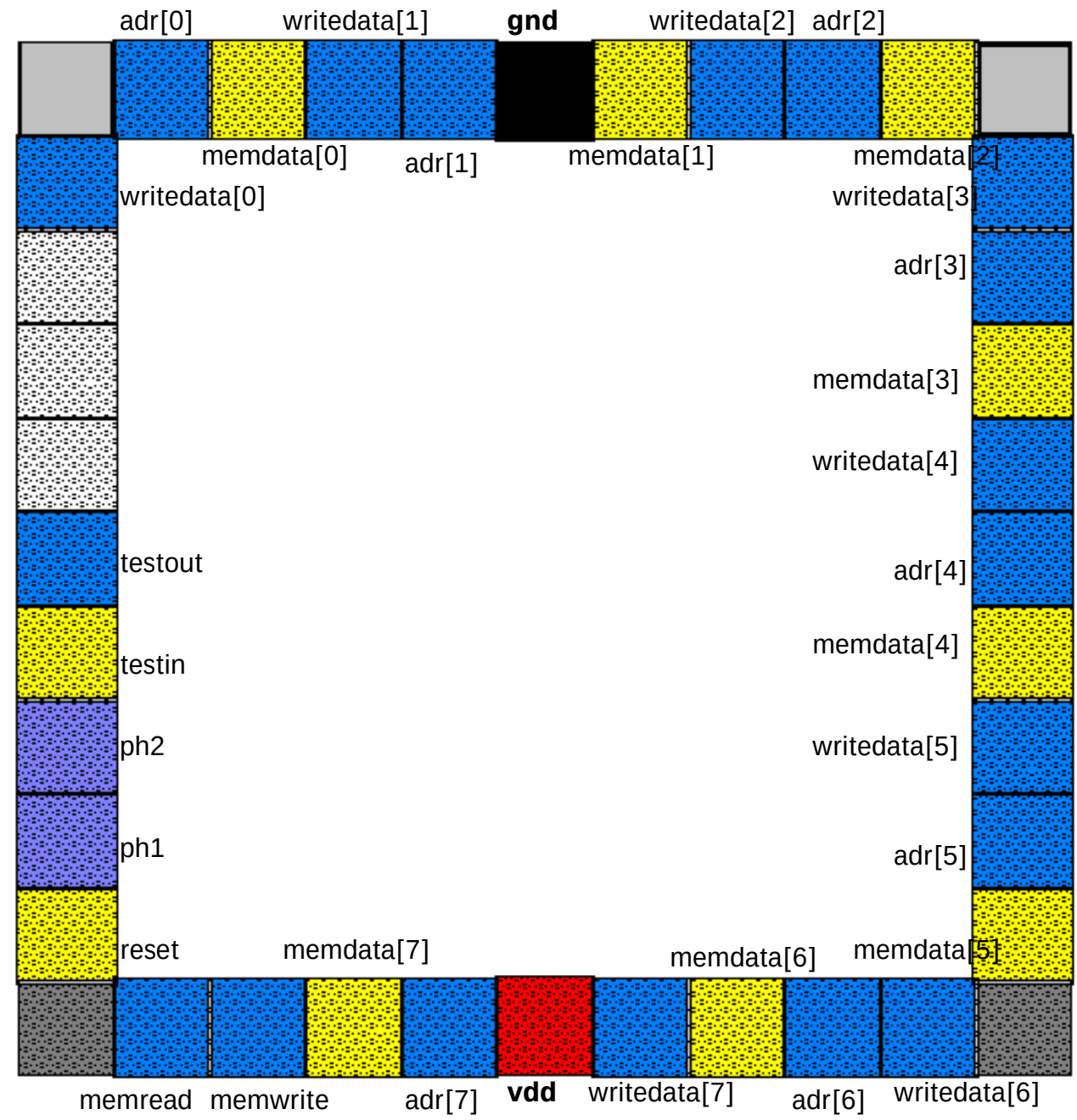
The Mini MIPS Microprocessor is an 8-bit microprocessor designed to support a limited subset of the MIPS instruction set. This design fits in the area constrained to a "TinyChip" MOSIS using a 1.5 μm process. In the CAD tool, the microprocessor's total area fits in a less than 2200 x 2200 λ square area. The microprocessor uses a two phase clocking scheme so there are two independent clock inputs, clock phi 1 and clock phi 2. Memory is stored off the chip, so there is several interface inputs dedicated toward interacting with memory. There are three 8-bit buses to memory, one is input for MemData, data from memory to the microprocessor datapath, another is out for Address to specify the address in memory that the datapath needs to read or write from, and the third is Write Data, the data that will be written to memory when required. Input Reset is used to reset the controller FSM to state 0 which is the initial state that begins every instruction, also it sets the memory Address to an initial starting position, 11111111. Finally, several pads are dedicated to power and ground. A chart of the inputs and outputs:

Inputs		Outputs	
ph1	clock phi 1	adr[7:0]	Specifies the memory address from which to read data or to write data to.
ph2	clock phi 2	writedata[7:0]	Sends eight bits to the memory at a specified address.
reset	Resets the controller to state 0, and sets the Memory Address to the starting position.	memread	Signal to the memory to read data at the specified address.
memdata[7:0]	Recieves 8 bits from the memory at a specified address.	memwrite	Signal to the memory to write data at the specified address.
testin	Check to see that the chips works from the manufacturer.	testout	Output of the test signal, should oscillate when testin is high.

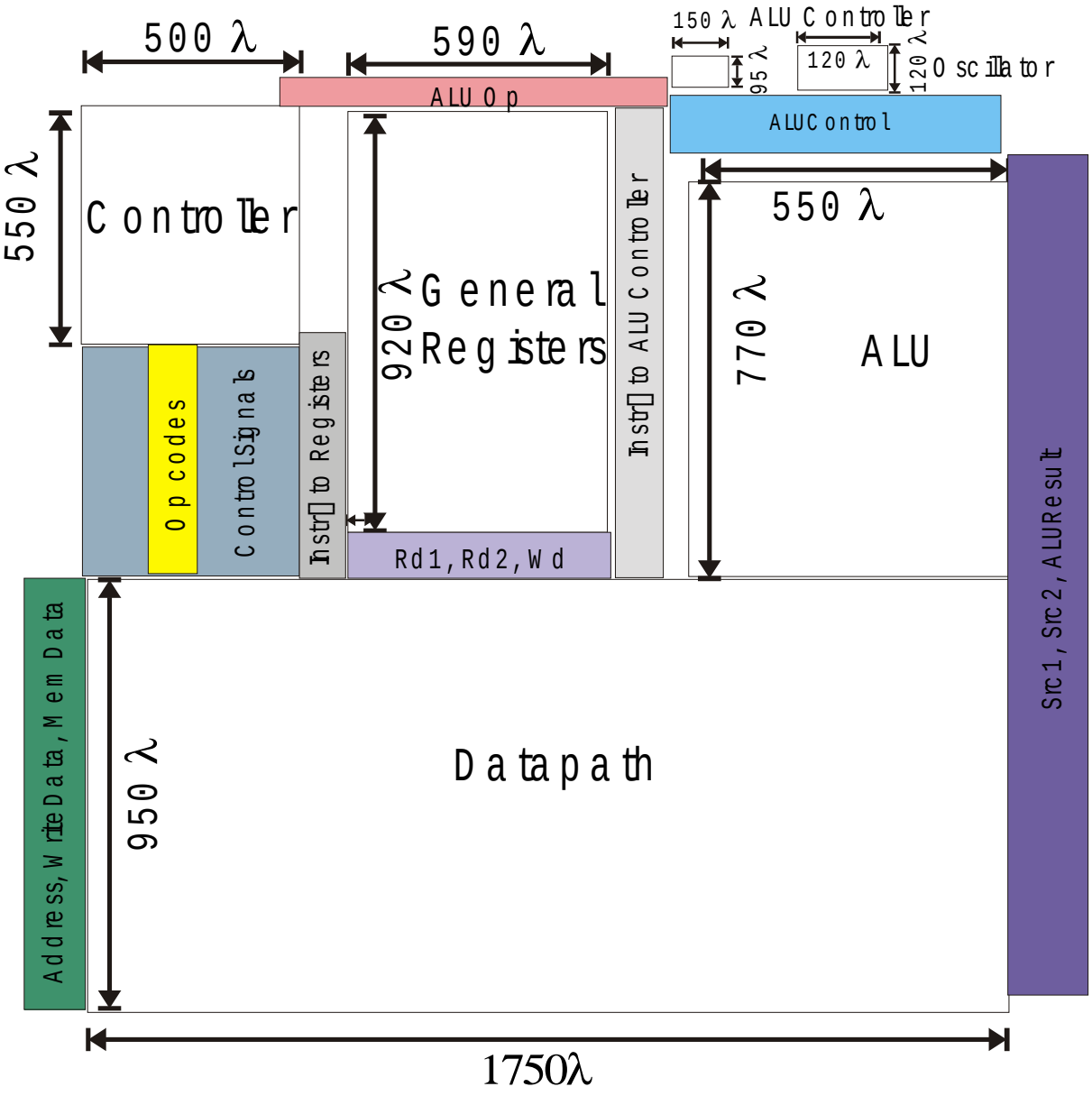
The microprocessor executes instructions which it fetches at an address specified by adr[7:0]. The instructions coming from the memory is interpreted by the controller, which is implemented as a finite state machine. This controller supports the following MIPS instructions with the standard MIPS encoding modified for an 8-bit datapth:

Instruction Operation	ALUOp	Funct Field	ALU control Input
load word	00	XXXXXX	010
store word	00	XXXXXX	010
branch equal	01	XXXXXX	110
add	10	100000	010
subtract	10	100010	110
AND	10	100100	000
OR	10	100101	001
Set on less than	10	101010	111

Chip Pinout


adr[0]
writedata[1]
gnd
writedata[2]
adr[2]
memdata[0]
adr[1]
memdata[1]
memdata[2]
writedata[0]
writedata[3]
adr[3]
memdata[3]
writedata[4]
testout
adr[4]
memdata[4]
testin
ph2
writedata[5]
ph1
adr[5]
reset
memdata[7]
memdata[6]
memdata[5]
memread
memwrite
adr[7]
vdd
writedata[7]
adr[6]
writedata[6]
vdd (input)
gnd(input)
output
input
clock (input)

Chip Floorplan



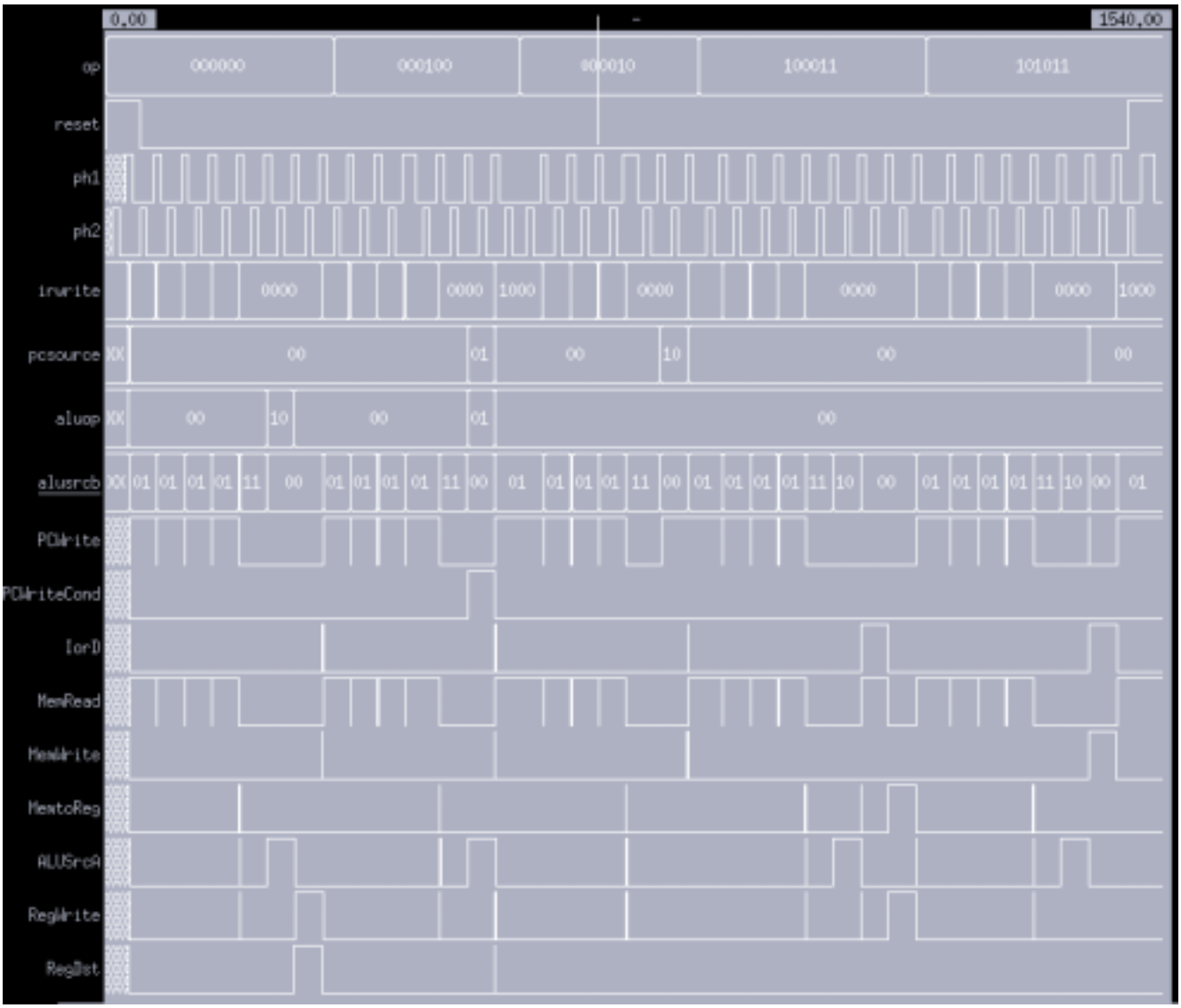
This is a block diagram of the major modules of the Mini MIPS, zerodetect is not included because it is small and fairly trivial. The major buses run from the datapath to the general registers, reading A or B from the registers and writing to the registers, and from the datapath to the 8-bit ALU that includes source 1, source 2 from two control muxes in the datapath and the ALU's result back to the datapath. Most control signals run from the main controller to the datapath, these wires spread throughout the top of the datapath to the zipper. Also, there are two register control signals from the controller to the general registers. The datapaht sends op codes to the controller for it to determine what the next state should be, these are the most significant six

instruction bits. Other instruction bits are sent to the general registers to control which register should be read from or written to. Finally the main controller sends ALU op codes to the ALU controller which in turn takes those op codes and some instruction bits to send the ALU control signals to the ALU.

Area and Design Time Data

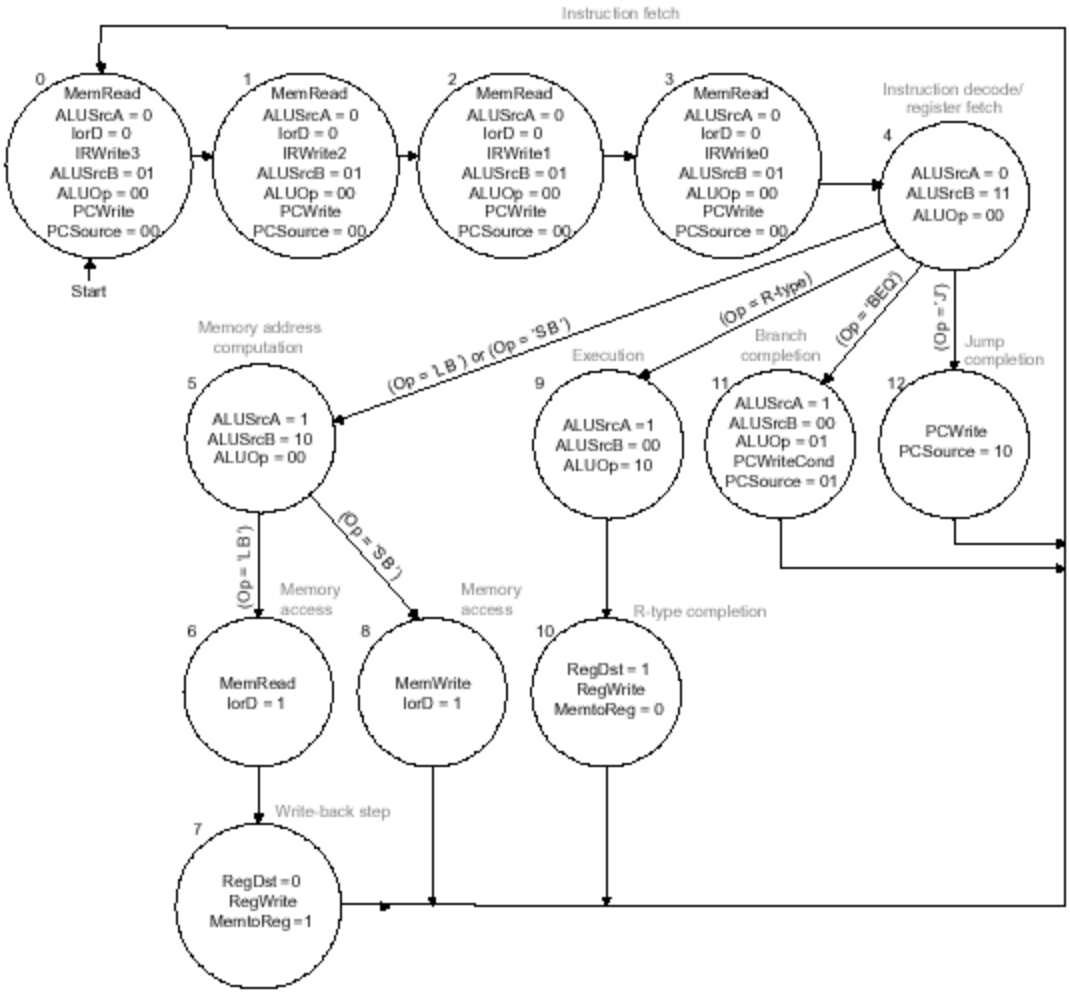
Cell	Type	Design Time (hrs)	Leaf Cell	Height (λ)	Length(λ)	Area(λ^2)	Notes
Mux2	sch	1	yes	80	35	2800	Modified to make modest
(modified)	lay	1					space savings.
Flop	sch	2	yes	80	96	7680	Saves nearly 40 lambda in
(modified)	lay	8					length from the original.
Register	sch	20	yes	48	70	3260	Implemented as a slice since
	lay	7					it is mirrored.
Decoder	sch	5	yes	450	90	40500	Has extremely long poly traces
	lay	7					which may be a problem.
PLA	sch	40	yes	550	500	275000	
(Controller)	lay	50					
Datapath	sch	10	no	950	1750	1662500	Uses modified Mux2s and
(modified)	lay	30					Flops.
Register	sch	3	no	390	650	253500	Mirrored to save space.
Array	lay	10					
General	sch	2	no	920	590	542800	Connects Register Array to
Registers	lay	20					Decoders appropriately.
8 bit ALU	sch	5	no	770	530	408100	
	lay	1					
ALU Control	sch	0	no	85	150	12750	
	lay	0					
Inverter	sch	0	yes	80	15	1200	
	lay	0					
Inverter 4x	sch	0	yes	80	20	1600	
	lay	0					
Buffer 4x	sch	1	no	80	50	4000	There are several modified
	lay	1					versions of this facet
Nand 2	sch	1	yes	80	20	1600	
	lay	1					
Nor 2	sch	0	yes	80	20	1600	
	lay	0					
weak pmos	sch	1	yes	590	20	11800	Responsible for driving psuedo
drivers	lay	2					nmos circuits.
nand 3	sch	1	yes	55	30	1650	Used in the decoders
	lay	1					
Zero Detect	sch	0	yes	950	50	770	Modifed since some of the
	lay	1					design rules changed.
Oscillator	sch	1	yes	120	120	14400	
	lay	1					

Simulation results



Controller

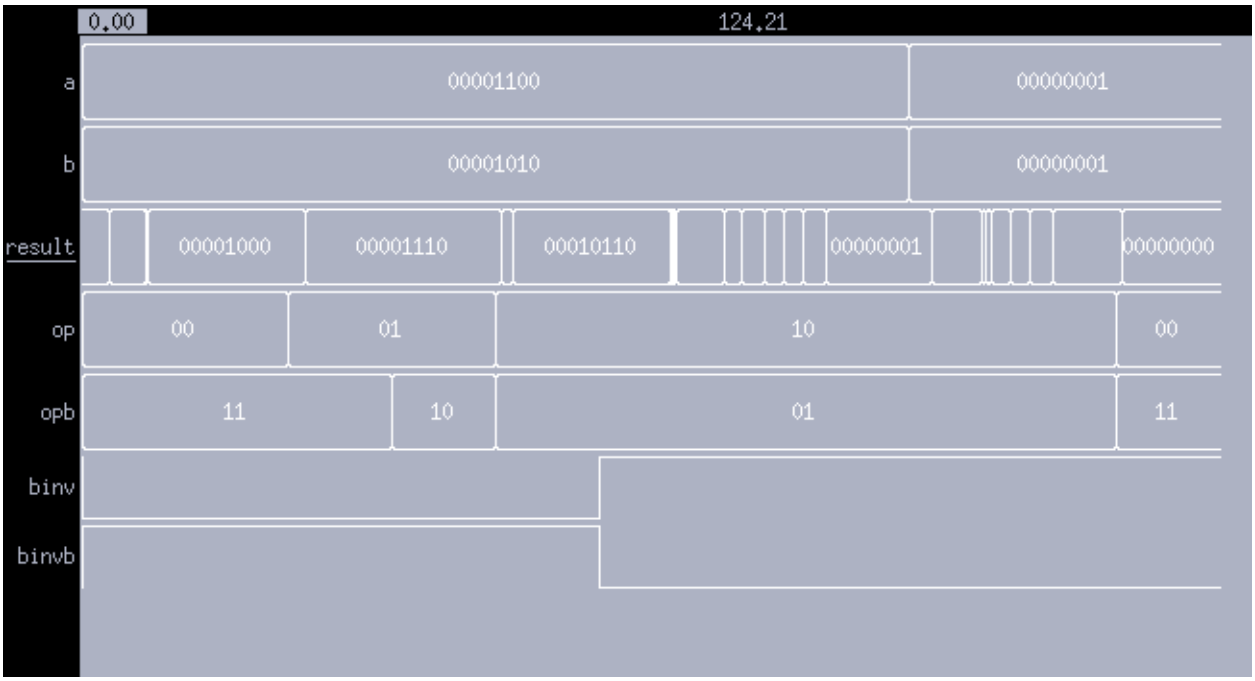
The controller for the processor follows the same FSM from Lab 4.



The controller utilizes a two phase clock, which is what cycles through each state in the FSM. The inputted Op codes will branch the machine to its respective state.

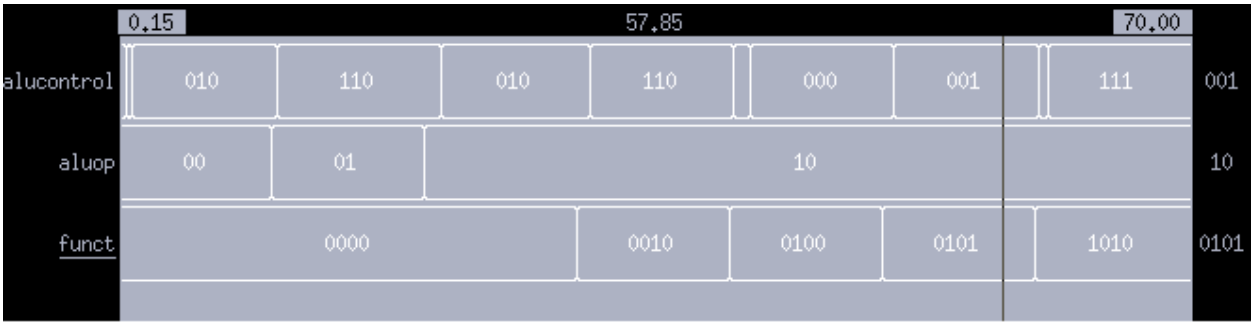
Instruction	Op Code
Load Bite	100011
Save Bite	101011
R-type	000000
Branch if Equal	000100
Jump	000010

Each Op code in the controller outputs the correct signals at the correct states.



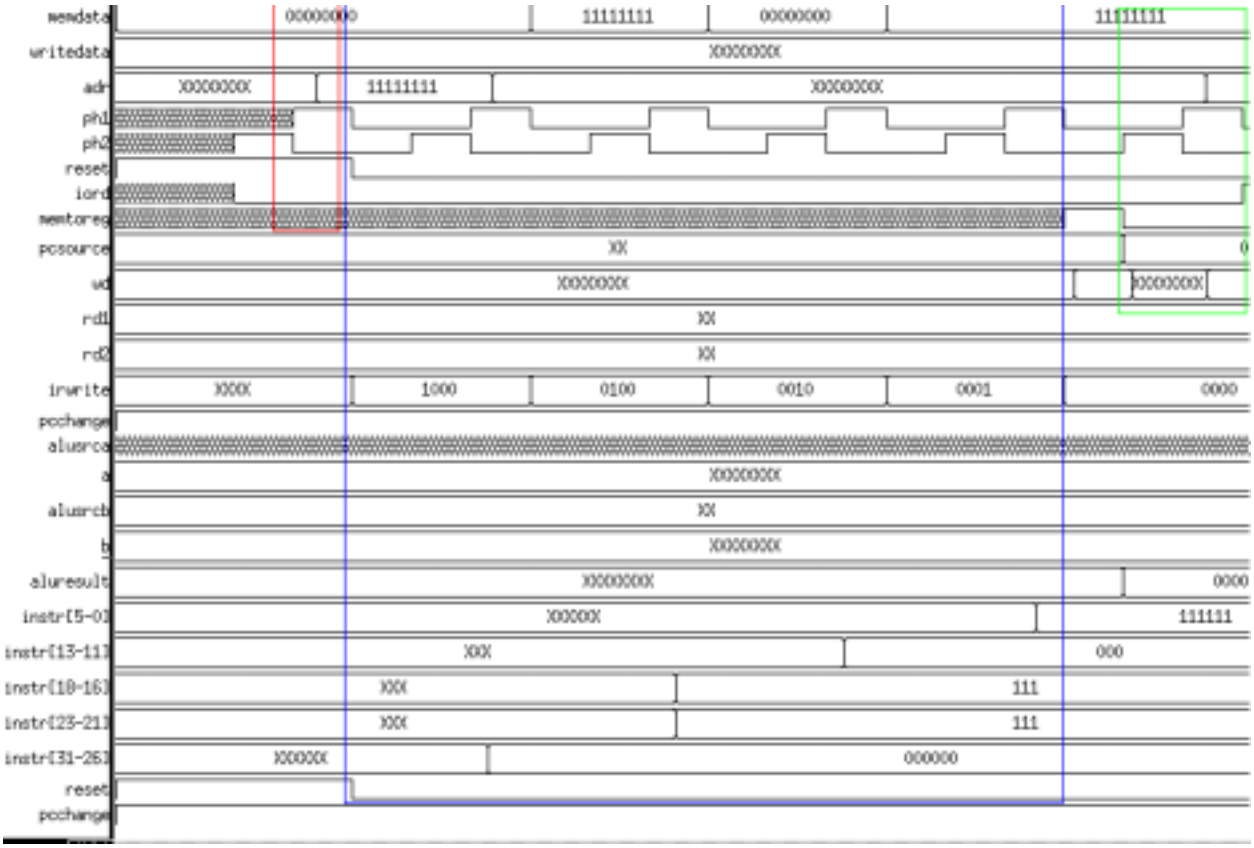
8 bit ALU

The ALU took a lot of time to simulate correctly, but finally the simulation worked. The ALU can AND, OR, and ADD two inputted signals. The op and binv inputs determine what function the ALU will perform. The simulation shows that the ALU AND's, OR's, and ADD's properly.



ALU Control

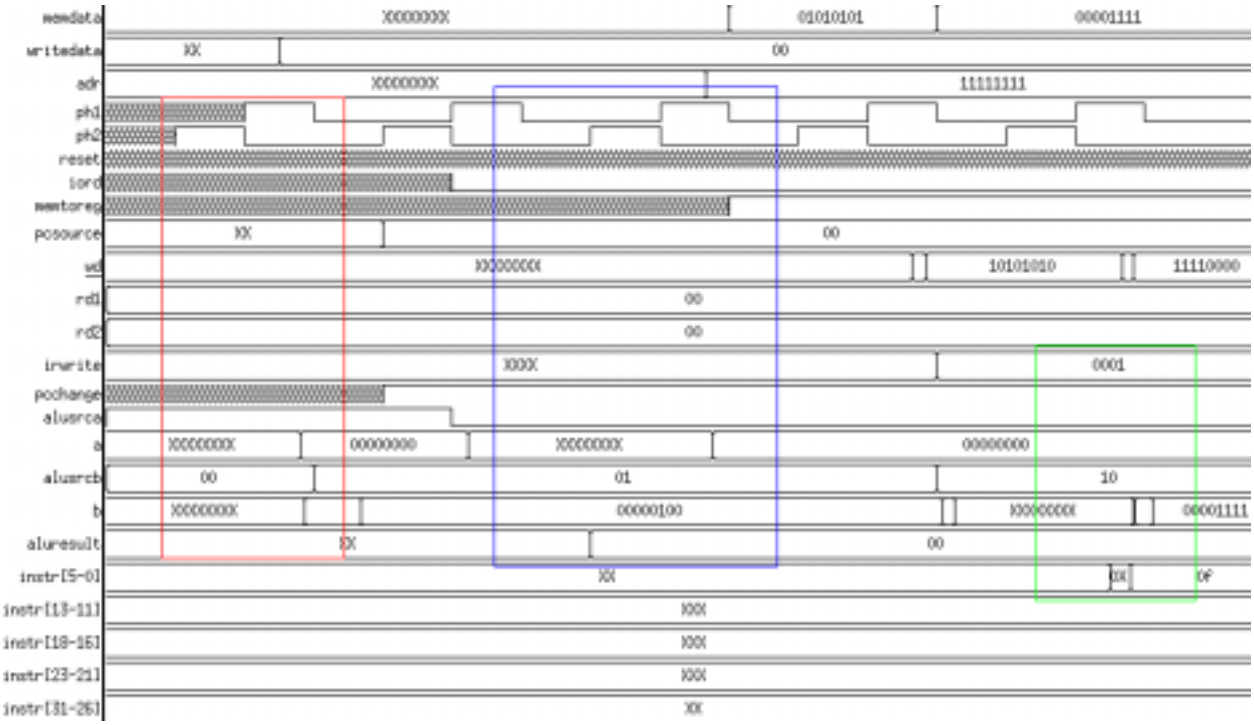
The ALU Control simulated accurately. The expected outputs was provided earlier in the report. The funct inputs are the four least significant bits in the instruction register. The aluop inputs come from the controller and the alucontrol signals go to the eight-bit ALU.



Datapath

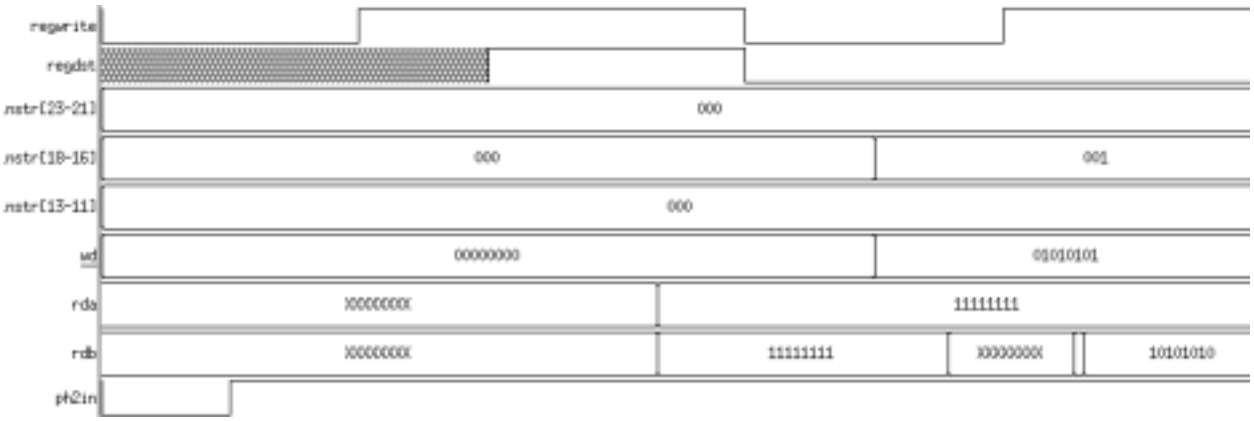
The first action is the activation of reset. Reset sets the adr to 11111111, but this only occurs if pcchange is activated after a cycle of ph2 and ph1. This is outlined in the red box. Irwrite enables the instruction registers to take on data from memdata. The highest bit in irwrite corresponds to the highest 8 register bits. In the simulation irwrite is activated from 1000 to 0001 in order, this indicated by the blue box.

Wd, write data to the registers is activated here by high MemtoReg, wd takes on the value from memory data register, which receives its input from memdata. The output wd is shown in the green box.



Registers A and B can take their values from inputs rd1 and rd2. The outputs a and b represent the first and second argument to the ALU. These outputs are selected by alusrca and alusrcb. Here both muxes are selecting the variable register values and these are outputted to the ALU, outlined in the red box. The second argument, b, can also be 4 if selected by alusrcb as 01. The first argument can also output the value of the PC register, which had acquired a value of 00000000 from ALU result from the PC mux. This is outlined in the blue box.

The second argument can output a value directly from the least significant eight instruction bits. This is shown in the green box, where the instruction registers take the value of 00001111 from memdata. When alusrcb selects the proper control, then the instruction bits are outputted.



General Registers

The General Registers are to write when regwrite and ph2in is high. Initially, no data values are loaded in the registers. By setting regdst high, the write address is selected from instr[13-11]. This is shown writing 00000000 to register 0. The reads are inversed from the written byte, this negation is acceptable because the data sent to the registers is already negated from the datapath. Nothing can be written unless both regwrite and ph2in are high. When regdst is low, the writing address is decoded from instr[18-16] which is written to register 1.

Verification results

The top level MIPS microprocessor has passed all DRC, ERC, and all NCC checks in the Electric CAD tool.

Test plan

The design of the chip includes a ring oscillator, which will test to see if the chip is functional. When the input is turned high, then the output will oscillate high and low. If the input is turned low, there will be no oscillation. The oscillator will be the first test for the chip.

The tester will first set the reset high and go through a clock cycle to set the controller to the first state and to set adr to 11111111.

Name	Format	6 bits	3 bits	3 bits	3 bits	5 bits	6 bits	Example
add	R	0	2	3	1	0	32	add $1, $2, $3
		000000	010	011	001	00000	100000	
sub	R	0	2	3	1	0	34	sub $1, $2, $3
		000000	010	011	001	00000	100010	
and	R	0	2	3	1	0	36	and $1, $2, $3
		000000	010	011	001	00000	100100	
or	R	0	2	3	1	0	37	or $1, $2, $3
		000000	010	011	001	00000	100101	
lb	I	35	2	1	100			lb $1, 100($2)
		100011	010	001	00000001100100			
sb	I	43	2	1	100			sb $1, 100($2)
		101011	010	001	00000001100100			
beq	I	4	1	2	25			beq $1, $2, 100
		000100	001	010				
j	J	2	2500					j 10000
		000010						

Next, each instruction that the processor supports must be tested. Load bite and save bite will be the first instructions to be tested. In order to get the whole 32 bits in, the processor will need to write in the 8 bit instruction registers four times, starting with the most significant bit. The two phase clock will cycle through the controller. If it is correct, load bite should output the correct adr and make memread high. Save bite will provide the desired writedata address along with memwrite, telling the memory to write the data. It is important to test these instructions first since they will be used later to test other instructions.

The tester will then check the R-type instructions. The proper instruction register will be inputted. To check if the R-type instruction executed properly, the tester will need to perform a save bite instruction and receive the data from the registers the execution was stored into.

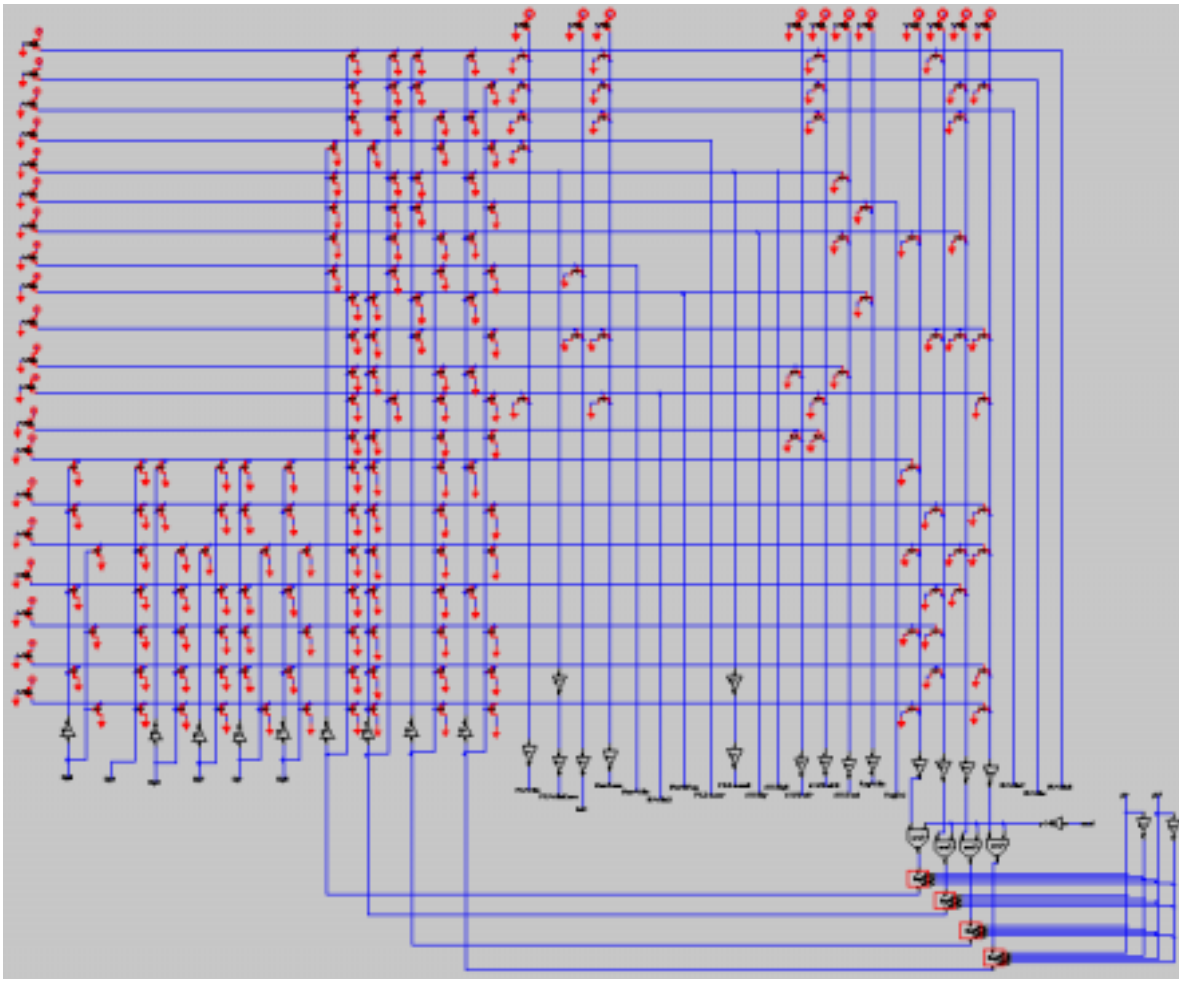
The branch and jump instructions will simply output the corresponding addresses that the instruction wants the processor to branch or jump to.

Finally, the tester will again activate reset and see if the outputs correspond correctly.

Note: the bits allocated for the registers in the table above is only 3 bits because the registers in this design only go up to 8, so 3 bits is sufficient. The table was extracted from the back cover of Patterson and Hennessey's book.

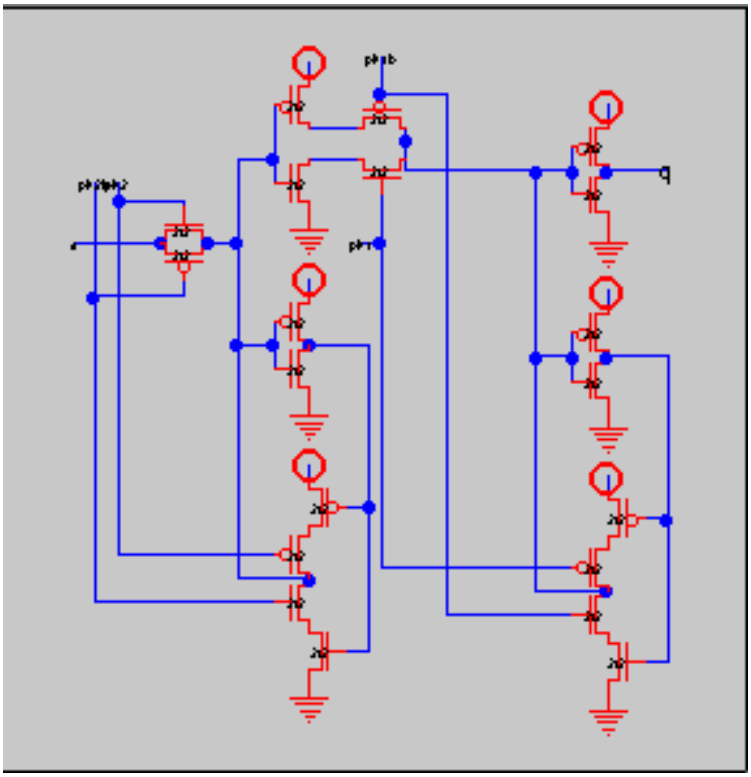
Complete schematics

Controller



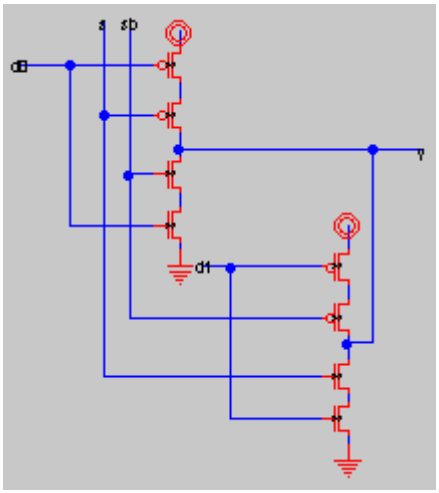
The controller was implemented as a PLA (Programmable Logic Array). The PLA used pseudo NMOS technology, placing only weak PMOS transistors and carrying out most of the design with NMOS transistors. The left half of the PLA is basically the ANDed portion of a PLA. Instead of using AND gates, DeMorgan's law was used and NOR gates were used with inverted inputs. The outputs of the ANDed portion are taken to the right half of the PLA, which is the ORed portion. This side is actually NORed and the output is inverted to give the correct value. The state outputs are fed back to the input. The output created is first ANDed with the reset inverted and each of those values are put into the flip flop. The controller uses a two phase clock. The output of the flip flops serve as the state inputs in the ANDed portion of the controller. Below is a table of the inputs and outputs.

Inputs	Outputs
Op[5] – Instr[31]	PCWrite
	PCWriteCond
Op[4] – Instr[30]	PCSource1
	PCSource0
Op[3] – Instr[29]	RegDst
	RegWrite
Op[2] – Instr[28]	IRWrite3
	IRWrite2
Op[1] – Instr[27]	IRWrite1
	IRWrite0
Op[0] – Instr[26]	Memread
	Memwrite
reset	MemtoReg
	ALUOp1
ph1	ALUOp0
	ALUSrcA
ph2	ALUSrcB1
	ALUSrcB0
	IorD



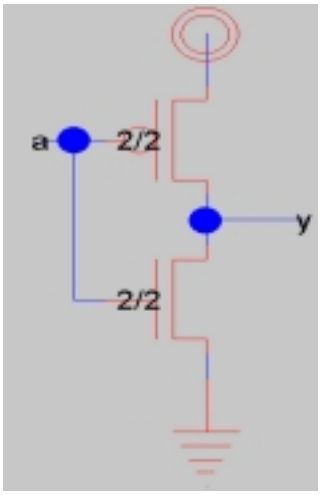
Flop

Schematic of the modified flop shows that it is entirely created on the transistor level to optimize spacing during layout. It is essentially two latches with one minor design difference. The original first latch inverter connection to the second latch's pass gate has been changed to a tri-state. This is done so the transistors can be put in series to save space. While the design is different it is logically the same since only pmos transistor are to pass high and only nmos transistor should pass low.

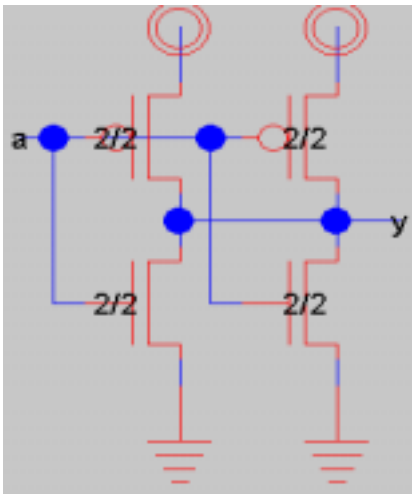


Mux2

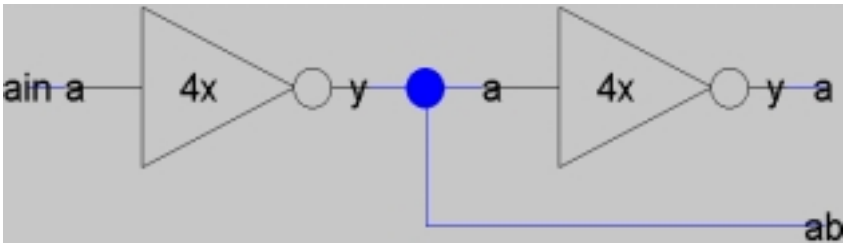
This is exactly the same as the Mux2 with two tristates only it is made with transistor so the layout can be further optimized.



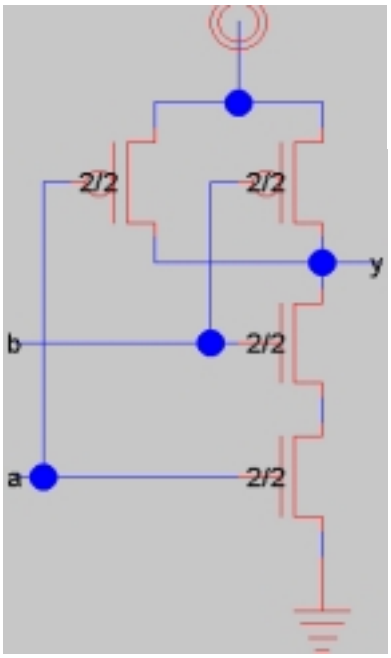
Inverter



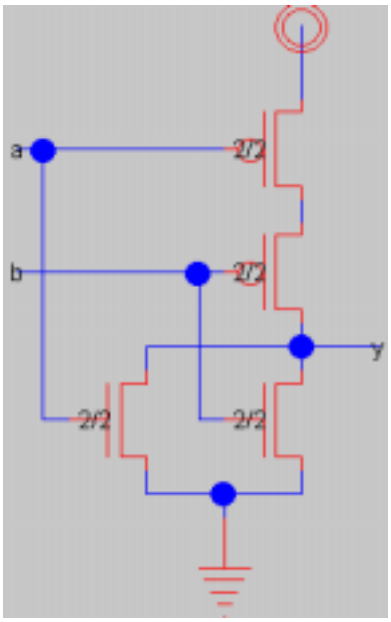
Inverter 4x



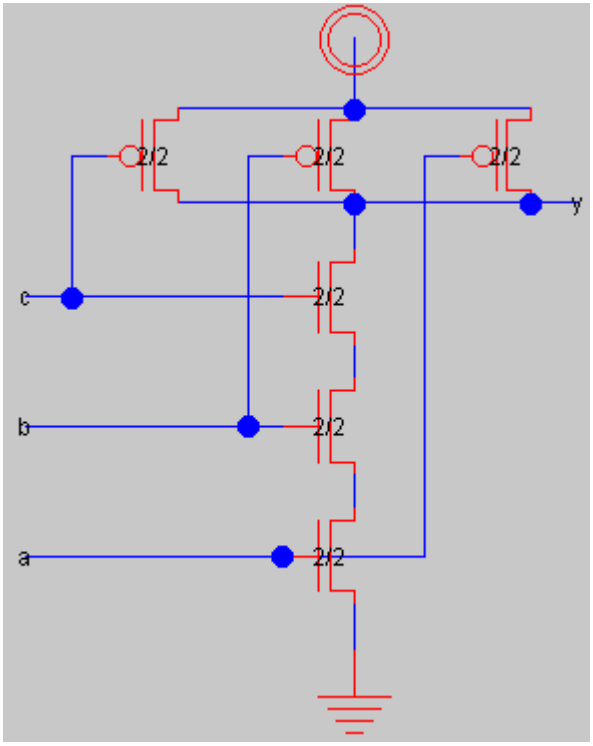
Buffer 4x



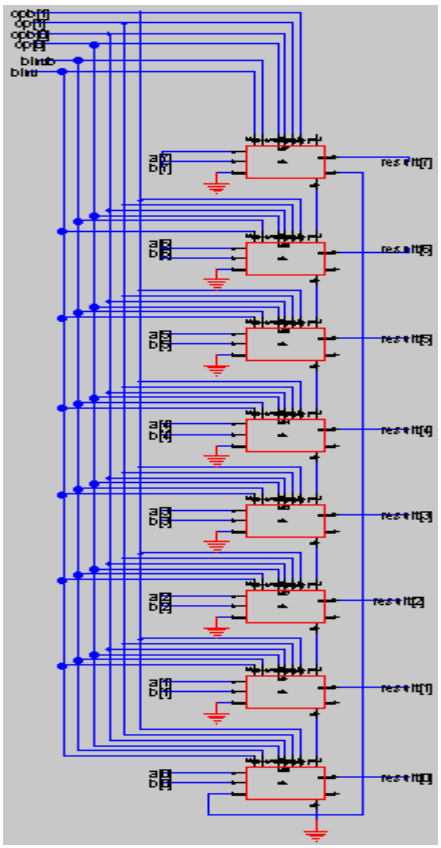
Nand2



Nor2

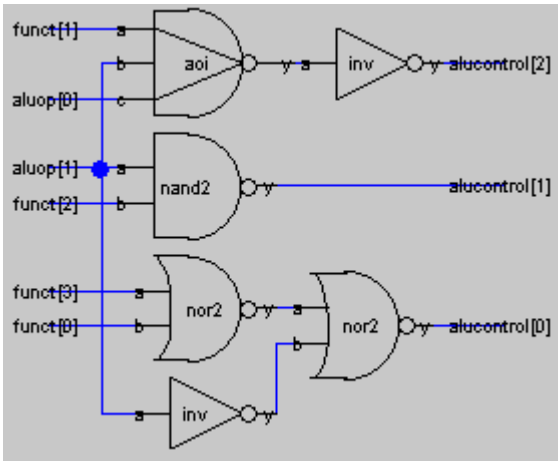


Nand3



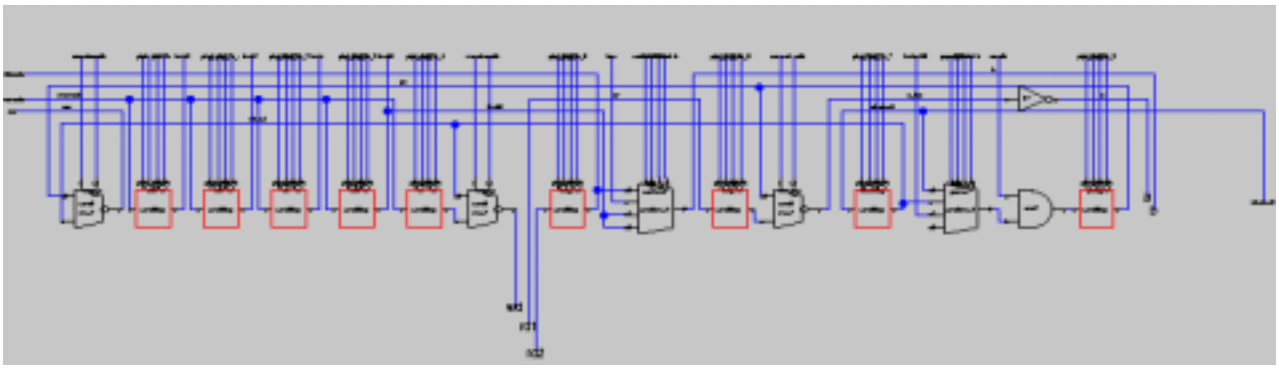
8 bit ALU

Schematic of eight ALUs connected with the lowest cin connected to ground. The set of the highest bit is connect to the less of the lowest bit to perform set less than operations. A bug was found and corrected in the ALU. The latest version of Electric (6.03 put up as 4/02/01) united all the set outputs that were not connected to anything. This caused an error in simulations. All the sets have been given their own names, this has corrected the error. The ALUs are taken from the lab 3 solutions library.



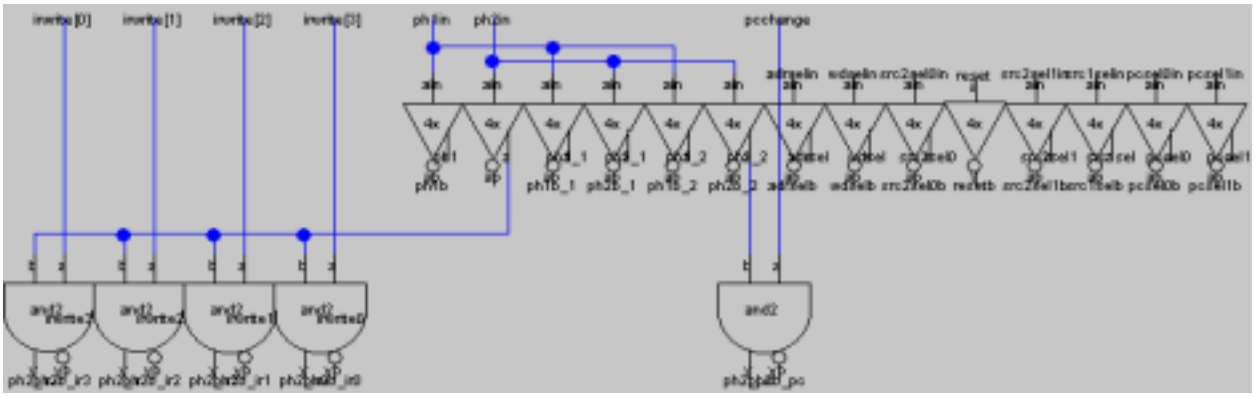
ALU Controller

The ALU Controller designed in lab 4. This facet has been taken from the lab 4 solutions. It takes two op codes from the Controller, aluop[1,0] and several instructions from the datapath, instr[3-0], to create control ALU control signals which are sent to the ALU.



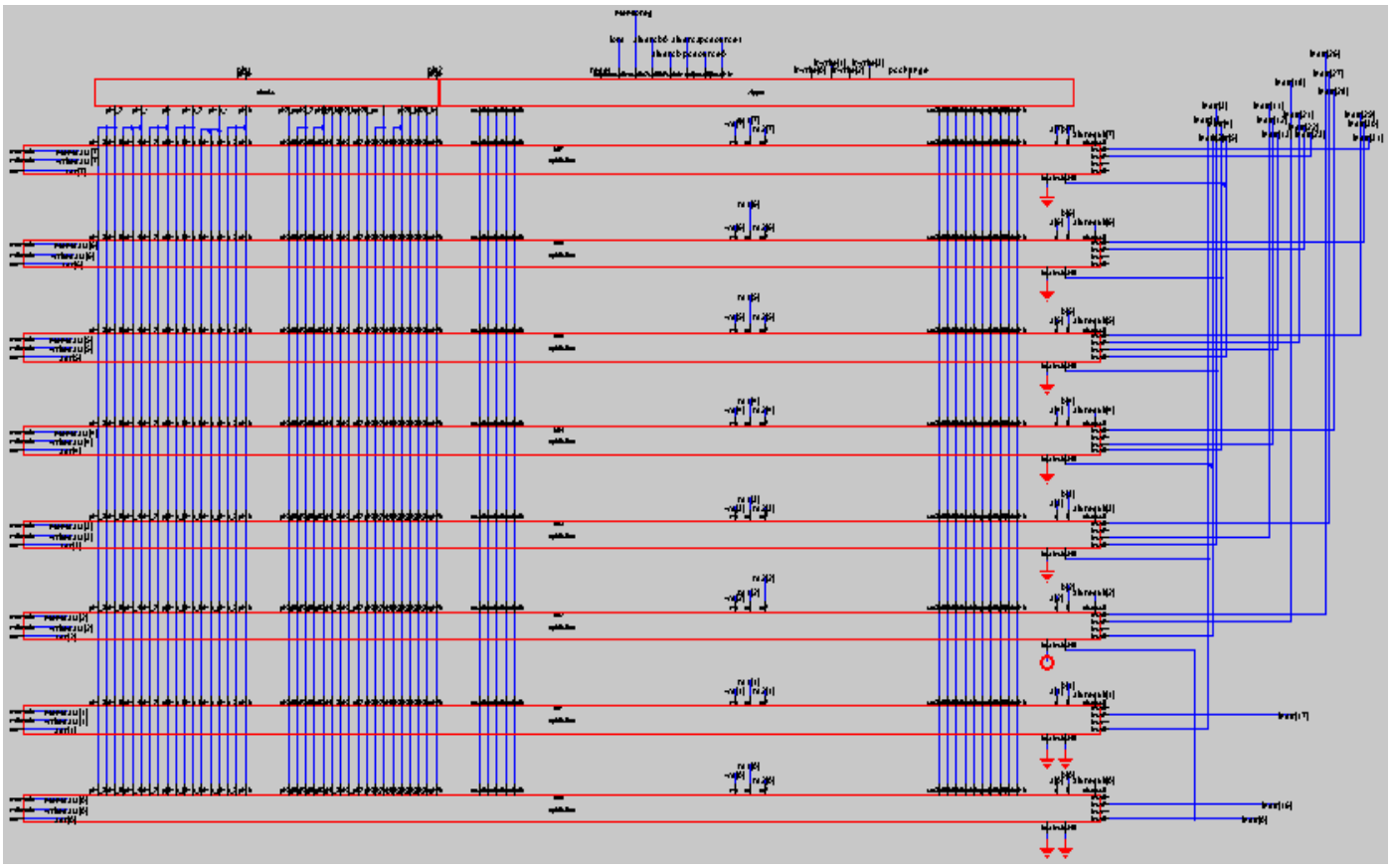
Datapath Bit Slice

The bit slice processes each bit on the datapath. It is composed of nine flops to slow data tokens and provide them appropriately, nine mux2s (some of which are combine for mux4s) to provide control and an and gate for reset capabilities. The flops are controlled by specific clock inputs which is sent by the zipper to each bit slice. Each flop is controlled by two clocks ph2 and ph1 and their complements, many of these are controlled clocks in that they do not send out signals every clock cycle. Inputs and Outputs to memory are sent to the external world through the pads, this includes the input Memdata, outputs Writedata and Address. The register slice has been removed from the original bit slice and I/O is provided to the General Registers. Also, the ALU has been removed, so several interface bits provide the two arguments, a and b that are outputs, and the input from the ALU, aluresult. Each instruction register, which is a flop, provides an output, although not all of these will be connected to other cells.



Zipper

The zipper receives signals from the Controller, buffers them, and provides true and complementary true to the datapath. Clocks ph1 and ph2 are inputs and buffered and sent as outputs to the datapath. Irwrite input signals along with the clocks provide signals telling when the instruction registers should take data. Some parts, including the decoders and buffers to the ALU have been removed from the original zipper since those parts have been removed from the datapath bit slice.

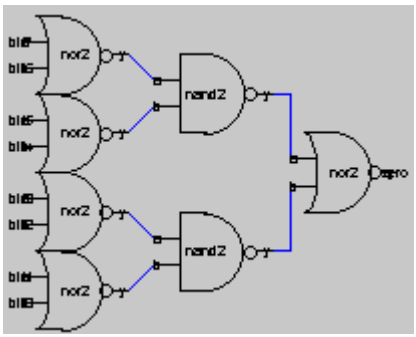


Datapath

The zipper is on top and takes control signals from the Controller. It then outputs the appropriate signals to the bit slices. The clock inputs for the flops are located on the left and the control signals for the muxes are on the right. Proper instruction signals are tapped out on the right hand side. Memory (external) I/O is on the left, the register I/O is in the center and ALU I/O is on the right. The instructions are also attached to the instruction shift inputs of the proper bit slices to allow the datapath to shift the instruction by two bits. The inputs four are connected to grounds and power to allow the datapath to directly add four when necessary, this is four program counter incrementing. A table of datapath inputs and outputs:

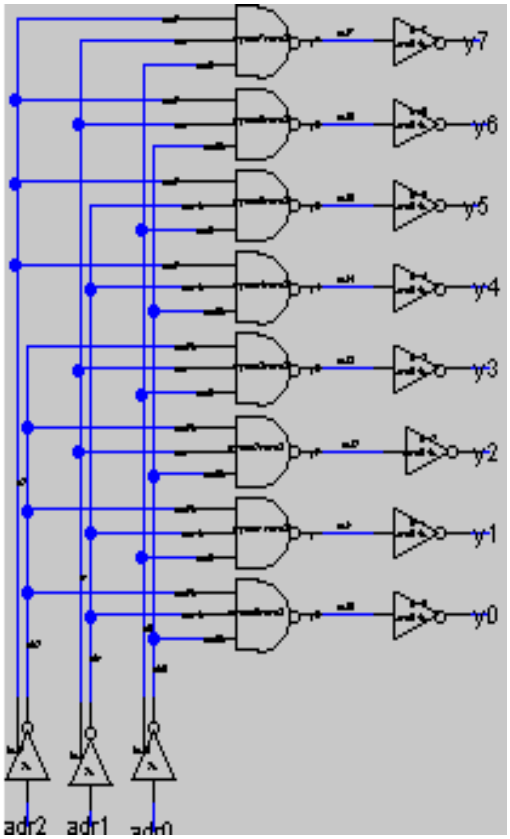
Name	Type	Connection	
memdata[7:0]	input	external	Retrieves input data from the external world.
adr[7:0]	output	external	Specifies the address at which to retrieve data.
writedata[7:0]	output	external	Sends data to the external world.
aluresult[7:0]	input	ALU	Gets the result from ALU calculations.
a[7:0]	output	ALU	Sends the first argument to the ALU.
b[7:0]	output	ALU	Sends the second argument to the ALU.
rd1[7:0]	input	registers	Gets data for the a register in the datapath.
rd2[7:0]	input	registers	Gets data for the b register in the datapath.
wd[7:0]	output	registers	Sends data to be written in the general registers.
alusrca	input	Controller	Controls the output a to the ALU.

alusrcb0,1	input	Controller	Controls the output b to the ALU.
iord	input	Controller	Determines where to select the memory address from.
memtoreg	input	Controller	Determines where to select the wd to registers from.
pcsource0,1	input	Controller	Controls the input to the PC register.
pcchange	input	Controller	Determines when to change the PC.
reset	input	external	Resets the address to some predetermined location.
ph1	input	external	Clock 1.
ph2	input	external	Clock 2.
instr[31:26]	output	Controller	Determines what state path to follow.
instr[23:21]	output	registers	Determines where to read rd1 from.
instr[18:16]	output	registers	Determines where to read rd2 from.
instr[13:11]	output	registers	May determine where which register to write to.
instr[5:0]	output	ALU Controller	Defines which ALU function to perform.



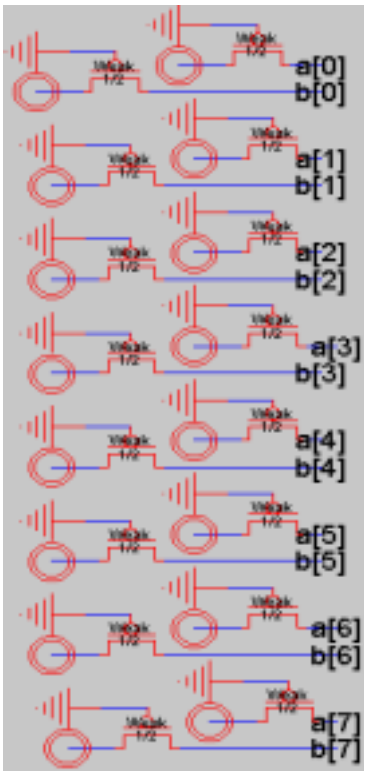
Zerodetect

Outputs whether or not all ALU results are zero. This is used for determining whether the program counter should change. It is taken from lab 3 solutions library.



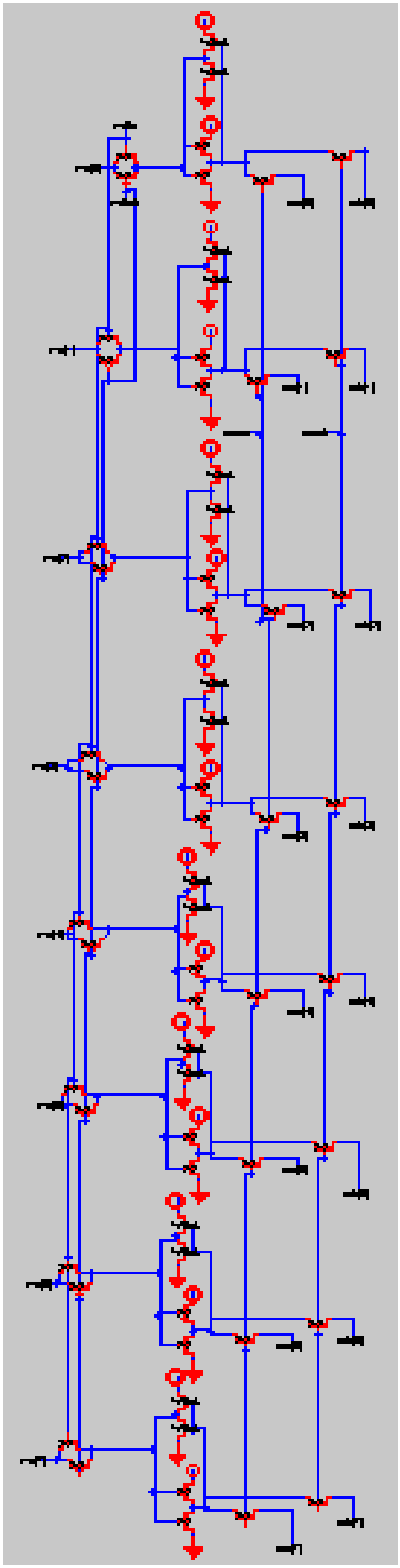
Decoder

Uses several nand3 gates to determine which bit should be active based on the three input bits. This is a three input decoder so it specify 2 to the 3rd bits or 8 addresses. The control signals are active high from the inverters at the end. Buffers to the signal strengthen the signal and provide true and complementary true. Note that each nand3 is implemented as an individual and unique facet even though they are exactly the same as all other nand3 gates. This is done because in layout each facet is sightly different to optimize space.



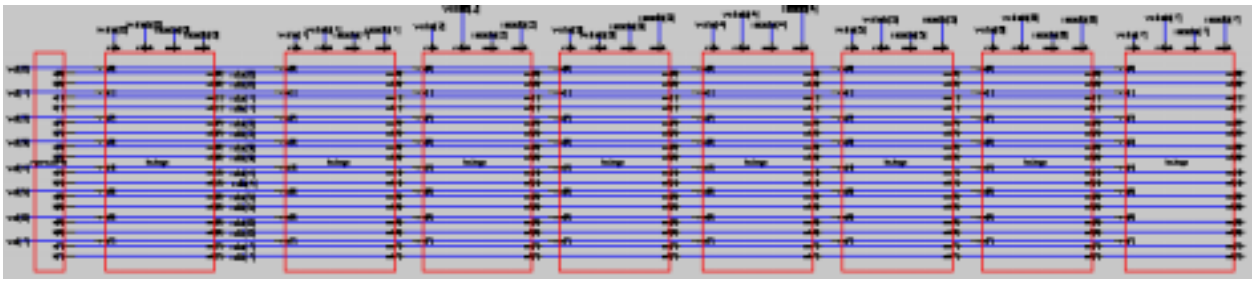
Weak pmos Drivers

These are used to drive the psuedo nmos parts of the register array.



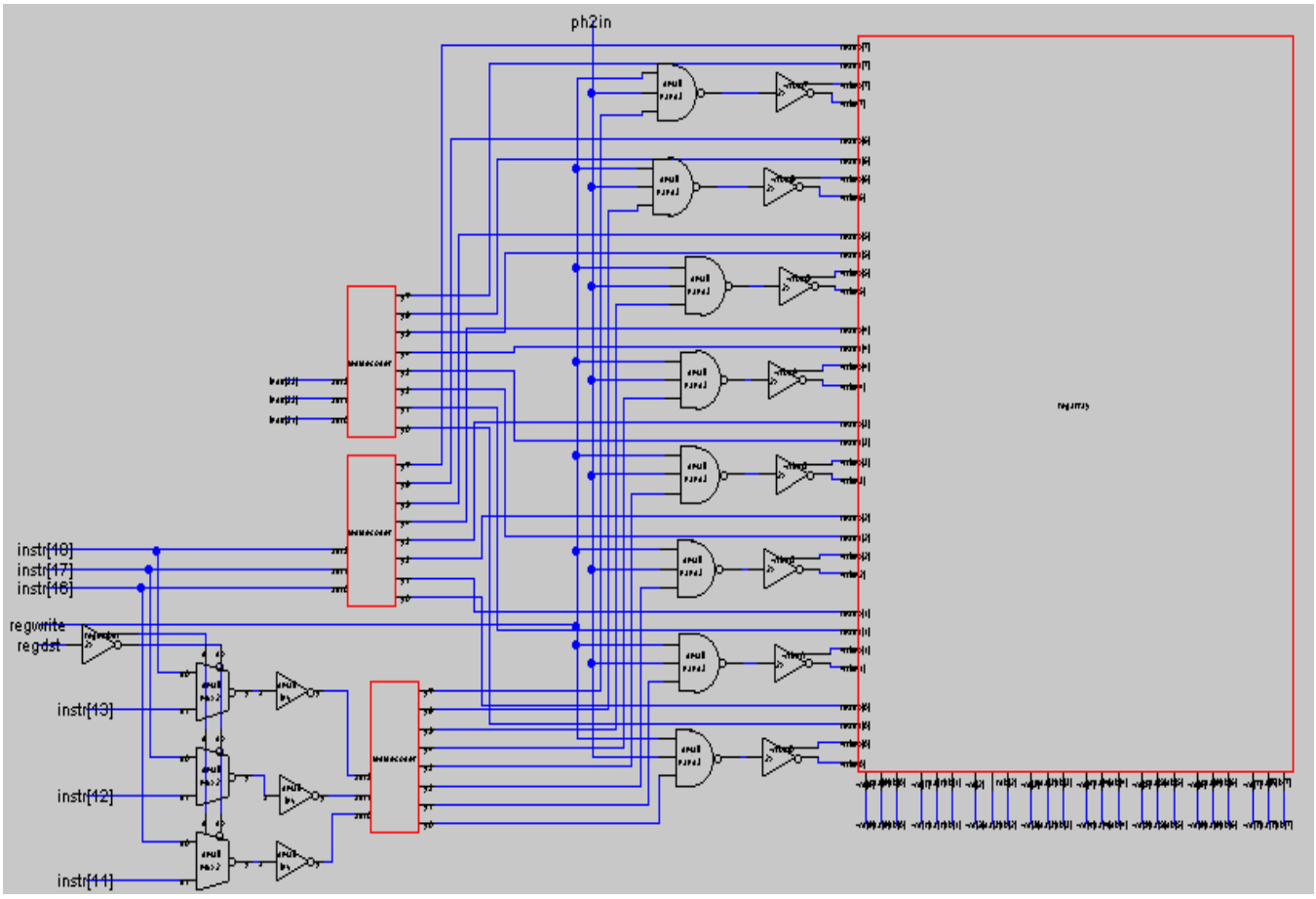
Register Slice

The register slice is a custom designed register cell. While these are larger and require more transistors than the six transistor static RAM, this is also easier to design, simulate, and guarantee correctness. This uses psuedo nmos technology and weak nmos and pmos feedback transistor. There is a “sticky latch” where an inverter is connected to a weak feedback inverter. The feedback inverter maintains the integrity of a value if nothing new is being written. However when the pass gate allows a bit through, then those pass gates must over power the feedback inverter. The outputs are connected only by nmos since there will be weak pmos drivers to put out high if the output nmos is turned off. Note that it is vital for nmos outputs to overpower the weak pmos drivers. This cell is implemented as a bit slice because the layout is mirrored.



Register Array

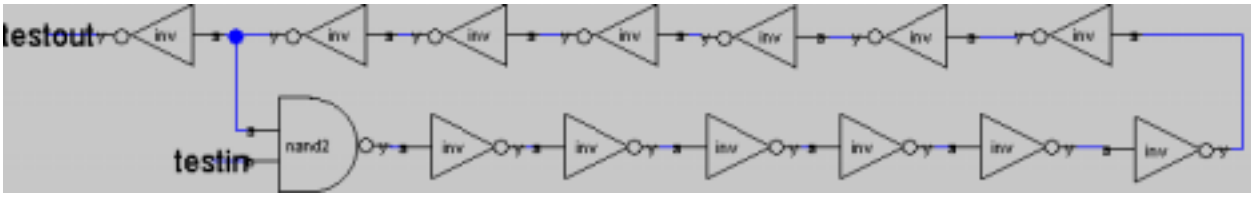
Combines all the register slices with the pmos drivers.



General Registers

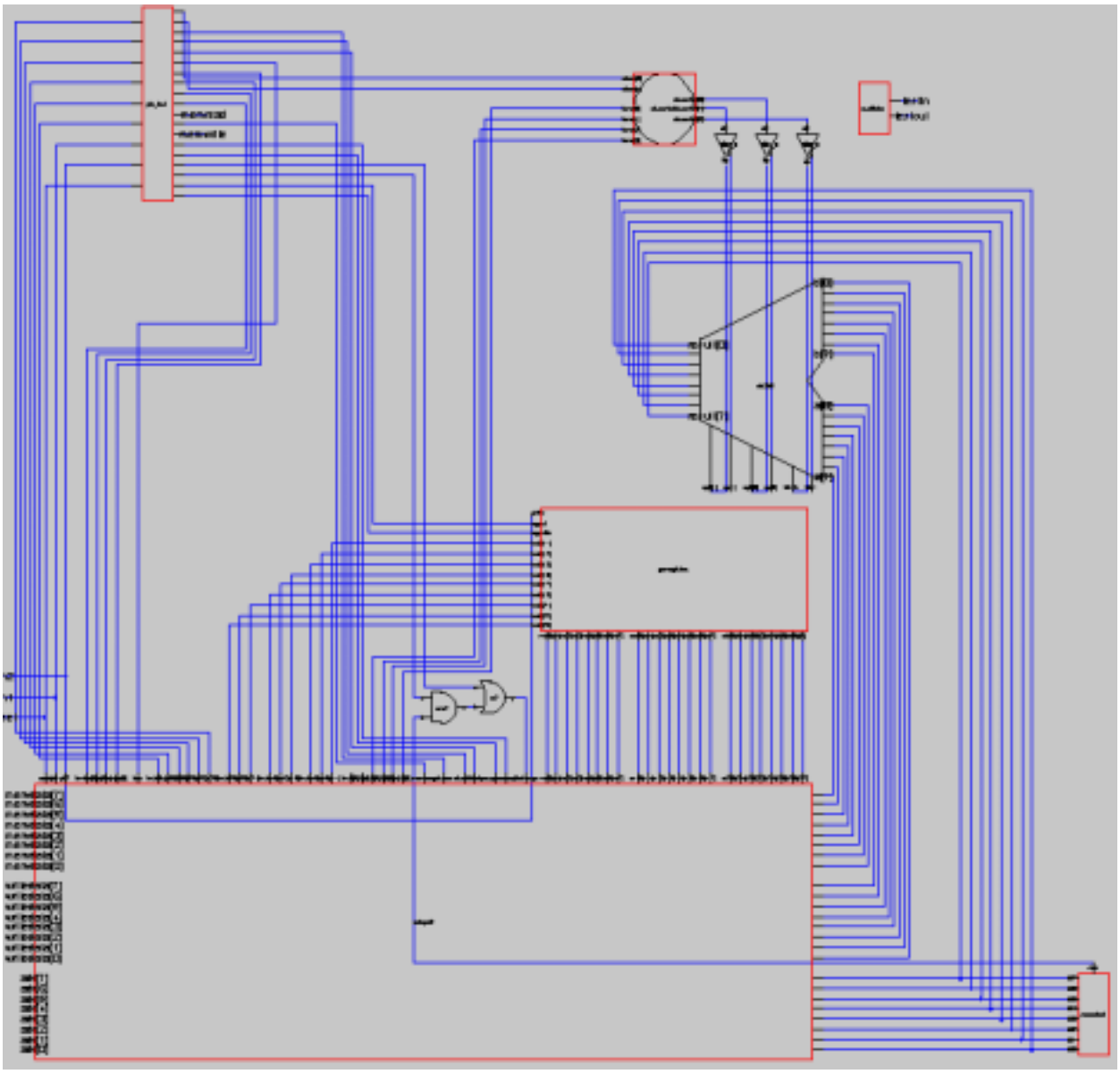
The Register array is composed of eight bit slices for a total of eight 8 bit registers and a series of weak pmos drivers driving each of the bit slices. There are three decoders, one for read 1, one for read 2, and the third for the write address. The read 1 takes instr[23:21] as inputs to determine where to read from, read2 takes instr[18:16]. Write register can be either instr[18:16] or instr[13:11] depending on the control signal regdst provided by the controller. Also, writing can only be enabled if regwrite is high, which is connected to the series of nand3 gates. The other input to the

registers is the clock ph2, so the registers can only be written when ph2 is high and regwrite is high. Buffers are used to provide a stronger signal, true and complementary true where they are needed. The bottom of the register array provides the I/O for the datapath where wd[7:0] is input from the datapath to write to the registers and rda[7:0] and rdb[7:0] are outputs to the datapath for registers A and B respectively.



Oscillator

The oscillator is used to test whether or not any manufacturing errors were made while making the chip. When the input testin is low, the output, testout, should also be low. To make initial tests of the chip, testin can be set high, which should force testout to oscillate. There is a single nand2 gate with twelve feedback inverters, which ensure an oscillatory response, and an extra inverter to the output. This makes for a simple test to begin chip validation.

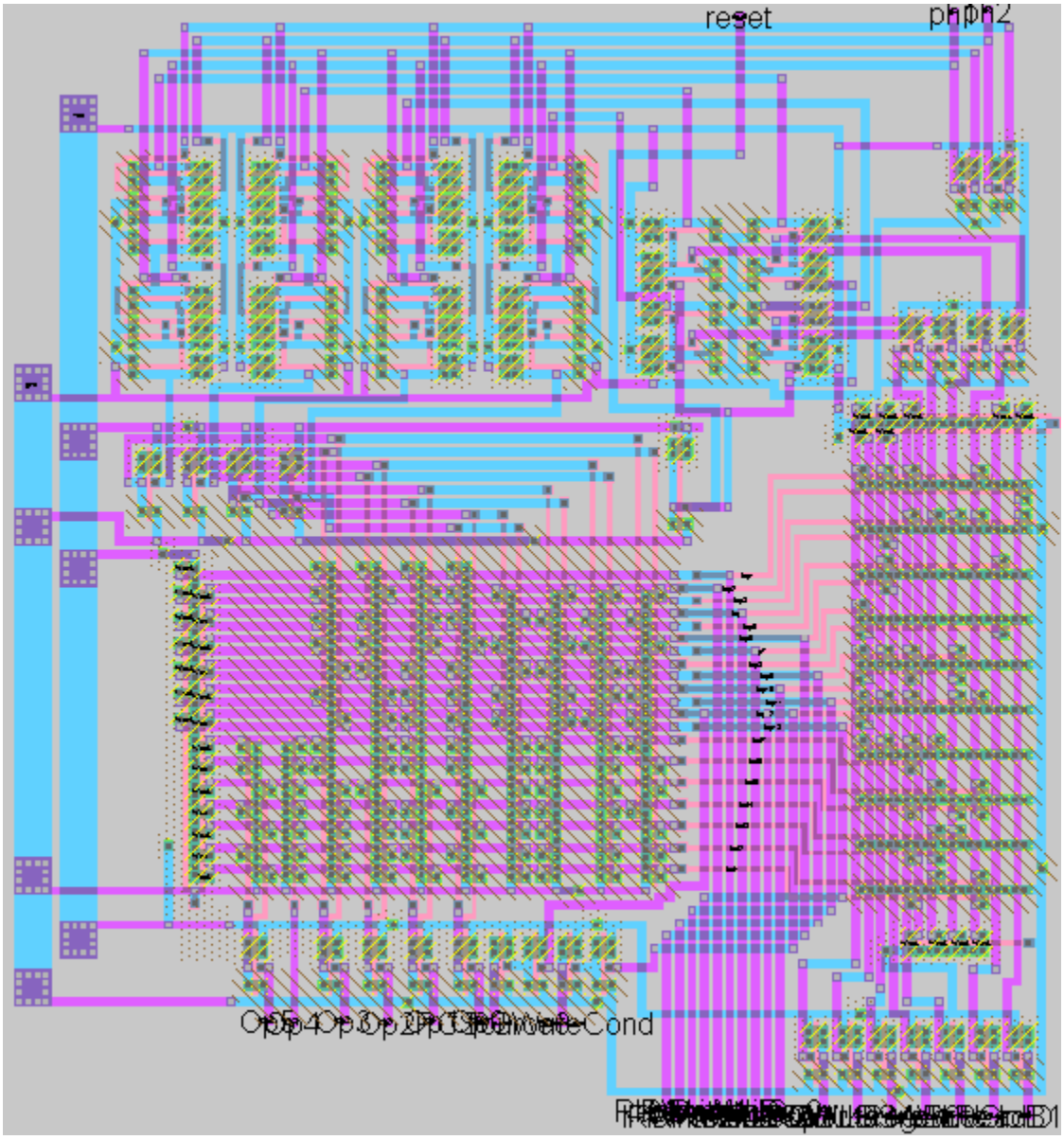


8-bit MIPS Microprocessor

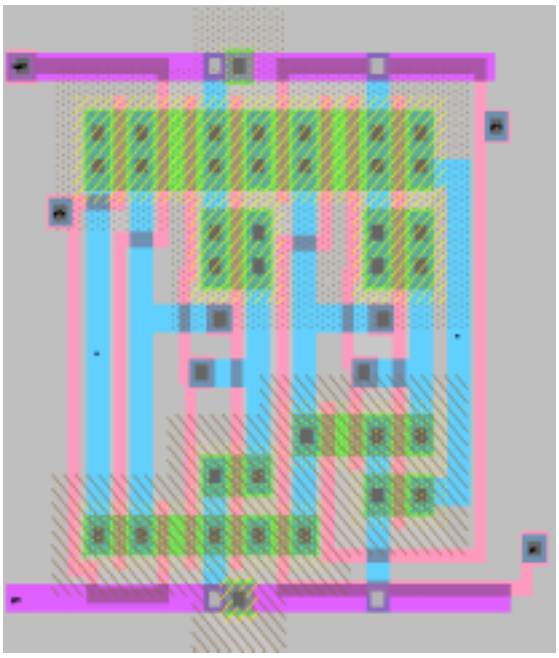
Top level schematic which shows the Controller in the upper left corner which sends control signals to the datapath on the bottom, to the ALU Controller in the upper right, and to the General Registers in the middle right. Most of the external I/O comes from the datapath on the bottom left. Other than the control signals, the major bus lines come from the interface between the datapath and the General Registers and the datapath with the 8 bit ALU. The zerodetect is seen here connecting to an OR and AND gate which combine with some Controller signals for controlling when the PC register should change. Other external I/O are clocks ph1 and ph2, input reset, and outputs memread and memwrite. The test oscillator is in upper right hand corner.

Complete layout

Controller

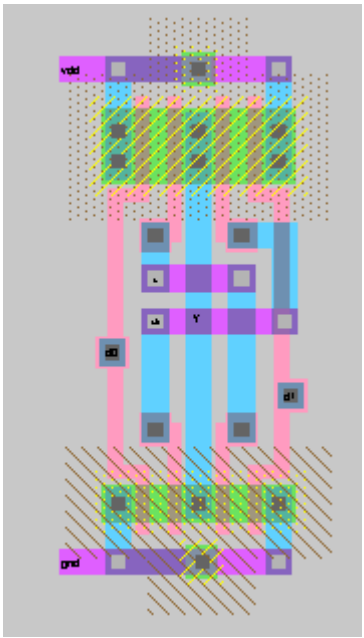


This is the layout of the controller. The column on the right is the ORed portion of the PLA, with the outputs tapped out at the bottom and the next states tapped out at the top. The ANDed portion is immediately to the left of the ORed portion. To reduce some height, the transistors forming the twelve states was folded over above the transistors forming the op inputs. The flip flops and AND gates are located at the top of the layout, feeding the state inputs above the ANDed portion of the PLA. The op inputs come from the bottom.



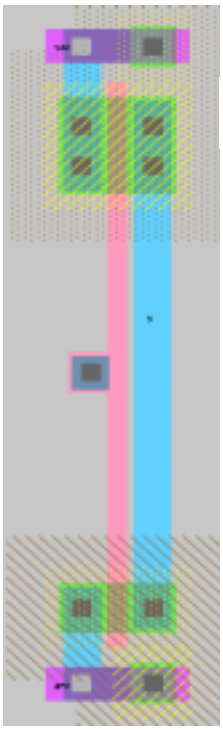
Flop

The modified flop is placed on an 80 λ pitch with metal 2 running horizontally and metal 1 running vertically. Several transistors are folded to allow for more compactness. This cell is optimized for area, and length in the horizontal direction. Control signals for phi 2 and phi 1 can be run vertically in metal 1. Input for d on the left side is meant to come in horizontally, but there is enough space to run the signal vertically. The output, q, comes out on the right side and can be run horizontally. This is one of the basic components of the datapath. Note that the two latches which form the flop are not identical so this is implemented on the transistor level. This passes DRC, ERC and all NCC checks.

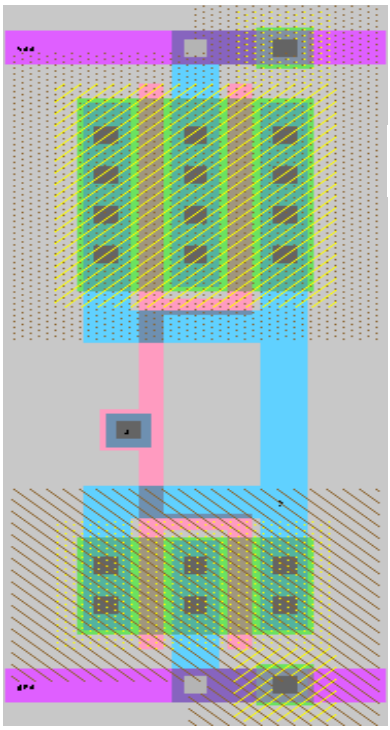


Mux2

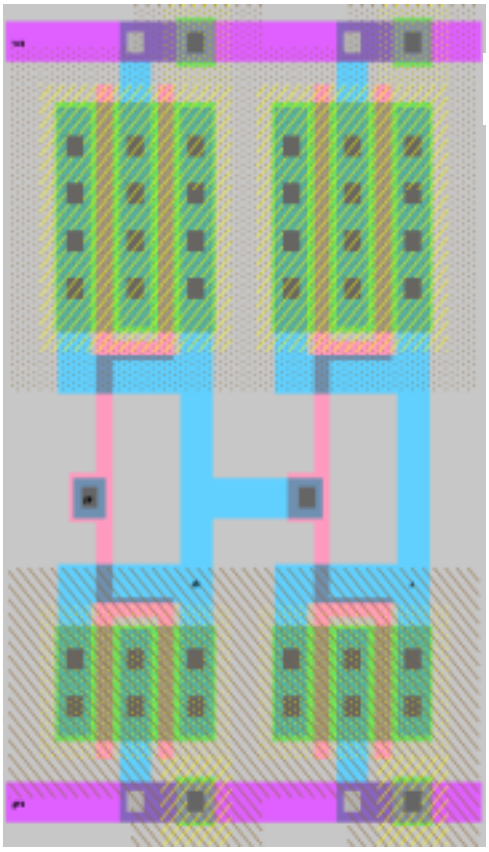
The modified mux 2 is nearly the same as the original, it is on an 80 λ pitch, with metal 2 running horizontally and metal 1 running vertically. The selection control signals are metal 1-2 contacts so the input can be run in with either metal 1 or 2. This is implemented on the transistor level to make a slight savings in space. This passes all checks.



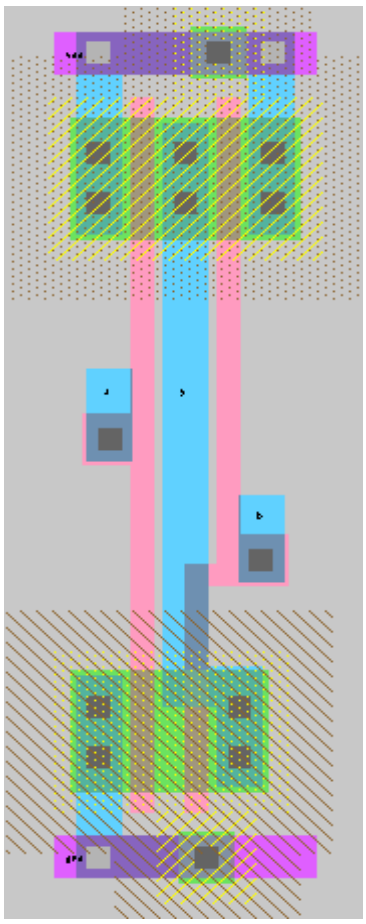
Inverter



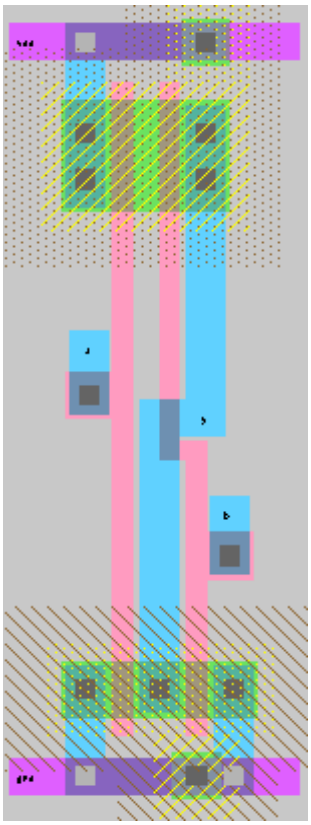
Inverter 4x



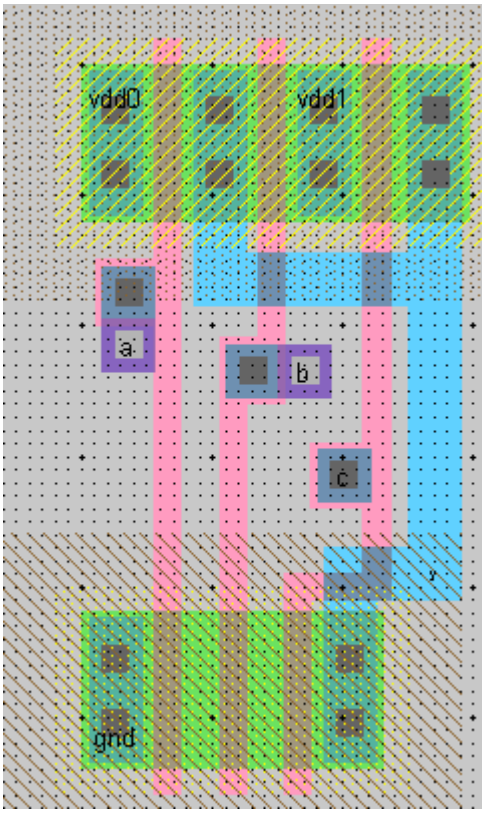
Buffer 4x



Nand2

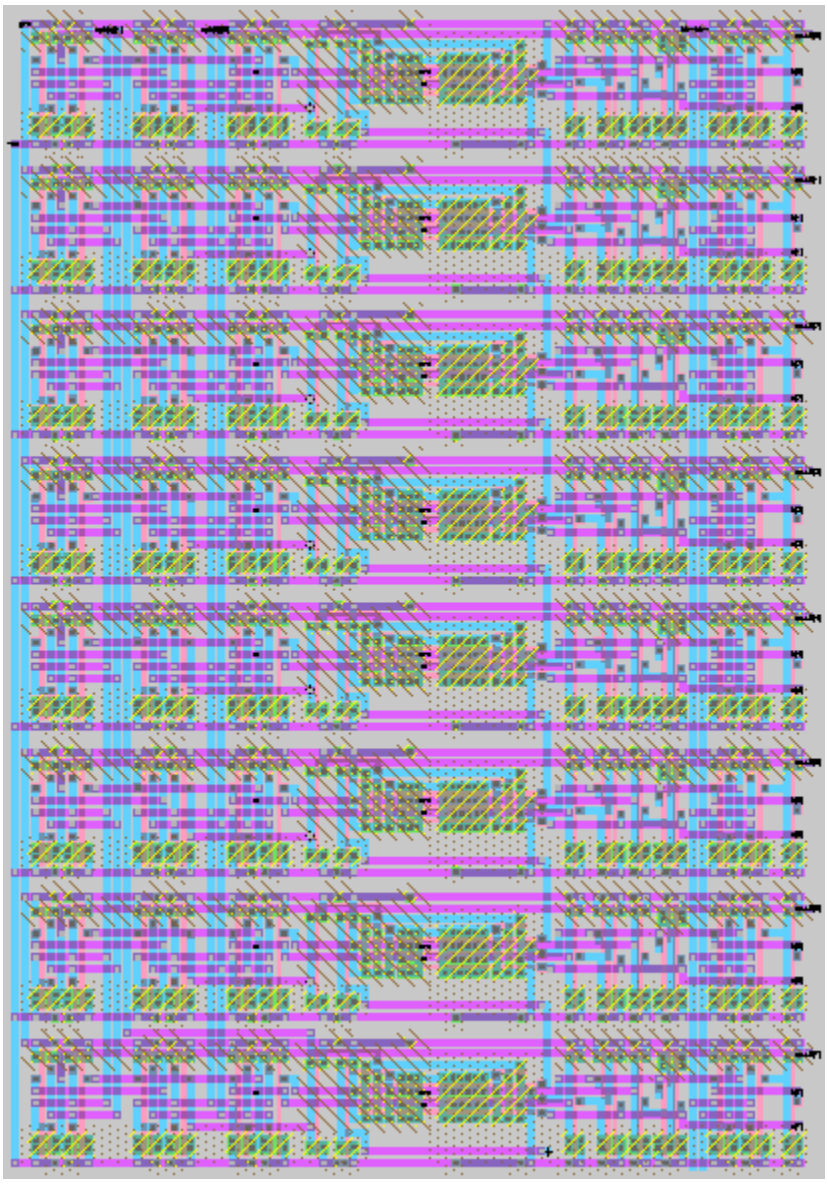


Nor2



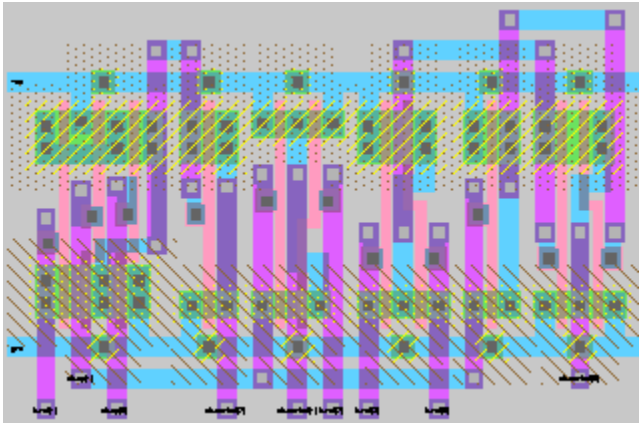
Nand3

Modified version of the standard Nand3. This is put on a small 55 λ pitch to minimize space requirements. These are extremely versatile because of their small size and lack of a great deal of metal 2.



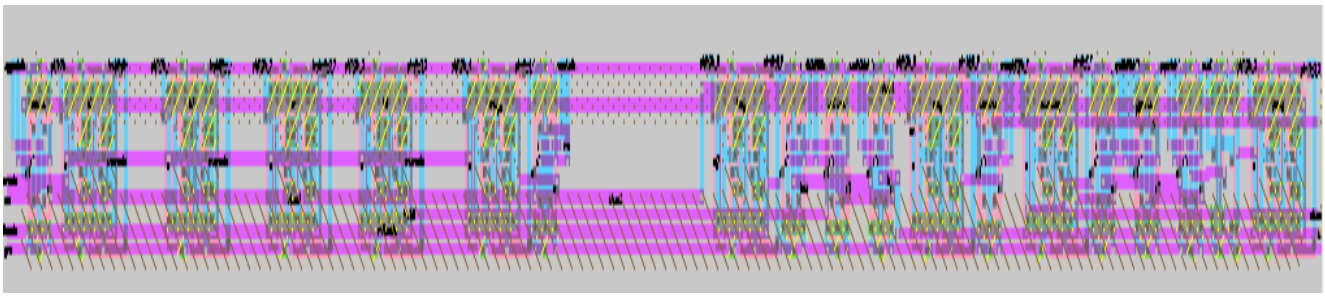
8 bit ALU

This is the 8 bit ALU it performs all the functions of an ALU, AND, OR, ADD, SUB, Set Less Than. The pmos transistors are on the bottom since this whole facet is inverse of the datapath. This allows more compact bus use between the datapath and the ALU. Inputs src1 and src2 are on the right. Output ALUResult is also on the right this is the interface with the datapath. Control signals are received from the top where they are sent from the ALU Controller. The ALU is taken from the lab 3 solutions library. It passes DRC, ERC and all NCC checks with the facet.



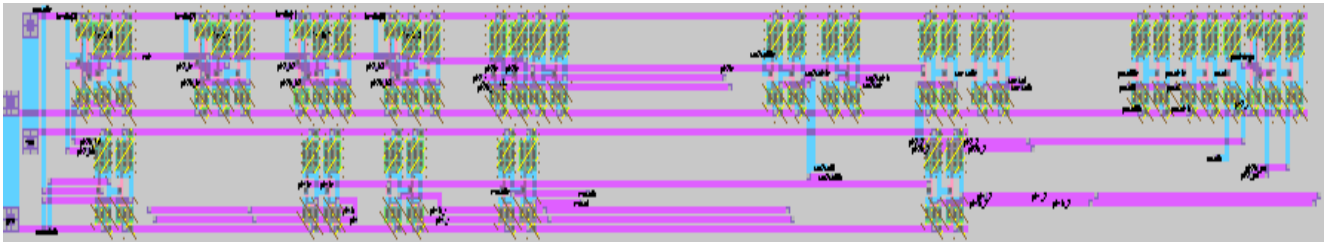
ALU Controller

The ALU Controller is also taken from the lab 3 solutions library. It uses standard layout of a 60 λ pitch and horizontal metal 1, vertical metal 2 wiring. It receives inputs on the bottom from the Controller, the and the datapath and sends outputs to the ALU on the bottom, the ALU Control signals. It passes DRC, ERC, and all NCC checks.



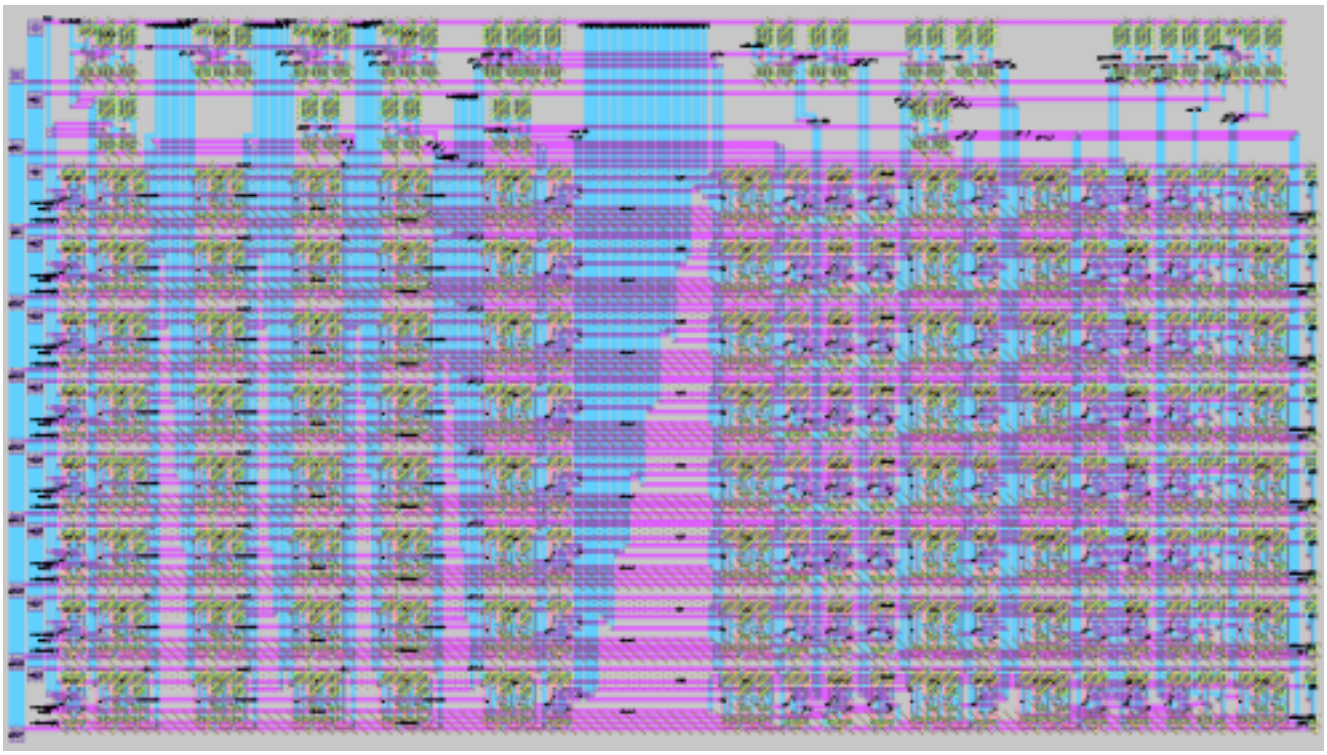
Datapath Bit Slice

Built on a regular 80 λ pitch, metal 2 runs generally horizontally and metal 1 vertically. Data is processed and regulated horizontally, external inputs and outputs to the outside world, MemData, Address, and WriteData are placed on the left. These will have to be connected to pads on the chip and they are all located on the right. Control signals are passed vertically in metal 1 from the zipper to all the slices. There are only two main leaf cells that compose the bitslice, mux2s and flops. These are highly optimized versions of the ones in class. There are nine flops, nine mux2s, one nand2, and two inverters in the path so even small savings in the leaf cells are magnified. The gap in the middle is left for I/O to the general registers. Instructions are tapped out from the instruction registers near the left side. These will be outputted to the Controller, ALU Controller, and General Registers. Outputs src1 and src2 are sent out on the right and input ALU result is received from the ALU on the right. The zerodetect facet will be added on the right side of the datapath. Power and Ground will run on the left side to unite all bit slices and the zipper. This passes all checks.



Zipper

Zipper is also built on an 80 λ pitch. This takes control signals from the Controller generally on the top, some are distributed. Instructions are sent through the top to the appropriate parts, the Controller, the ALU Controller and the General Registers. This is why there are so many gaps in the zipper. Metal 2 runs horizontally and metal 1 vertically. Outputs to the bit slices are passed through the bottom. The gap in the middle is to permit the I/O from the bit slices to the registers to go through. This will snap perfectly on top of the bit slices to form the datapath. All the outputs to the bit slices are perfectly aligned. However, inputs from the Controller are not because the Controller will send its signals to many different places, it is extremely important to leave gaps for metal 2 to transport the signals.



Datapath

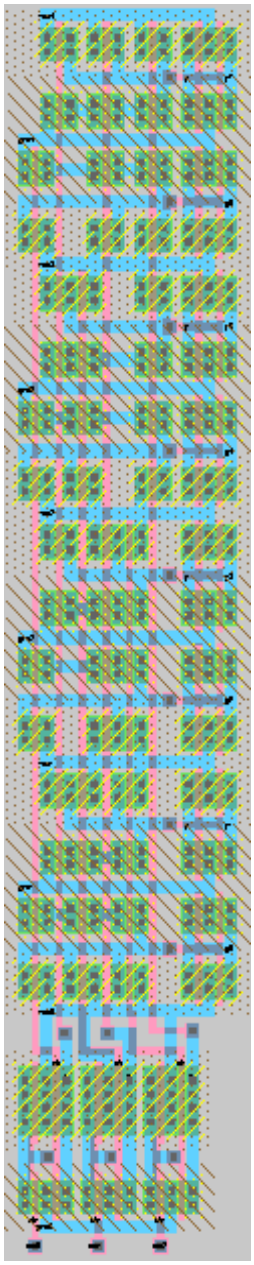
This is built from eight bit slices and one zipper on top of the bit slices which takes signals from the Controller, distributes, provides true and complementary true, and buffers signals to the bit slices. Control signals are sent vertically in metal 1 to the bit slices. Data is processed horizontally, generally left to right. The I/O for the Controller is distributed throughout the top of the datapath. I/O for the external world, the MemData input bus, Address and WriteData output busses. General Register I/O is sent up through the center of the datapath, input busses Read data 1, Read data 2, which receive information from the registers and output bus Write data which sends data to be stored in the registers. I/O for the ALU, sending two arguments and receiving the result is on the right side of the datapath. The instruction registers sends their data through the upper left of the datapath. This passes all checks.

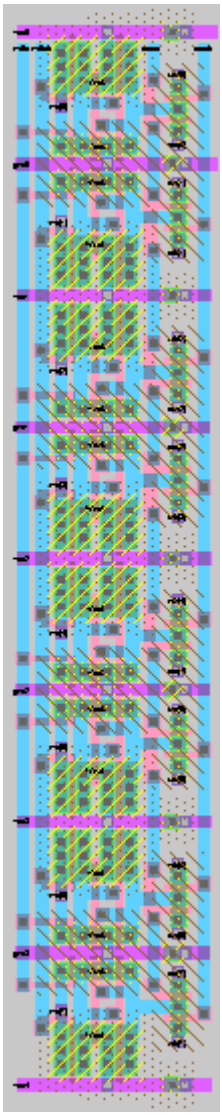
Zerodetect

The zerodetect is implemented on an 80 λ pitch with the signal passed upward. This is placed at the end of the datapath and interfaces with ALU result. The output is sent out on top in Metal 1 where it will interface with an AND, an OR gate, and some Controller signals to signal when the Program Counter should change. This passes all checks

Decoder

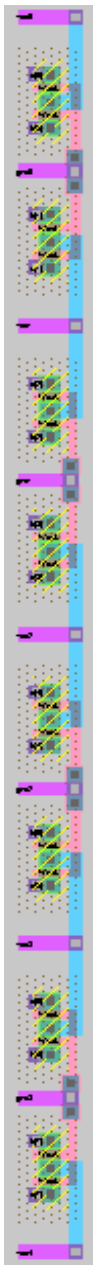
This is the decoder used to interpret the proper register address from instruction bits. There are buffers on the bottom to generate true and complementary true. Three address bits are received on the bottom and buffered through the decoder to generate read and write signals on the write. The true and complementary are passed up in long polysilicon traces, this could be a problem. Each part of the decoder is like a nand3 only with different inputs. Note that there is no metal 2 usage, this allows metal 2 wires to be ran completely over the cells. This passes all checks.





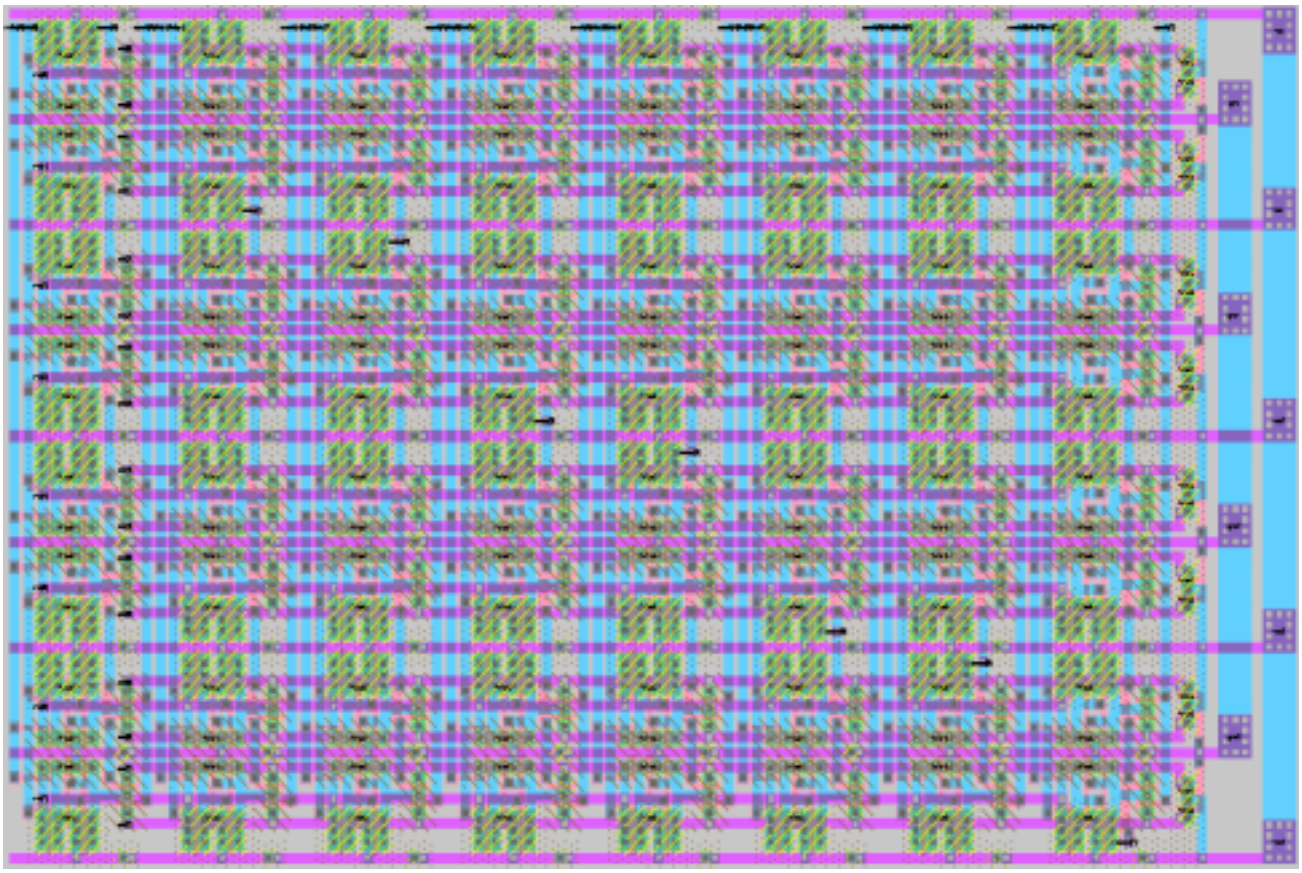
Register Slice

The register slice is a vertical slice of eight custom designed registered cells. This is implemented as a slice since all registers are mirrored. Metal 2 runs horizontally and Metal 1 runs vertically. Read and write control signals are passed vertically along a slice, while data is passed along horizontally. The read and write control signals will come from decoders while the data bits will come from the datapath. These registers use pseudo nmos technology, as well as weak transistor technology. This is why the regular pmos transistors are so large, to overpower a weak nmos transistor. Also, there are weak pmos drivers along each horizontal bit from which there are nmos transistors in each transistor to possibly over power them. Note that the read b control signal is wired in both polysilicon and metal 1,this is used to avoid metal 2 usage which would block horizontal metal 2 wires. These polysilicon parts are made extremely wide, 5 λ, to reduce resistance. This was used instead of the 6 transistor static RAM because of the ease of implementation. This passes all checks.



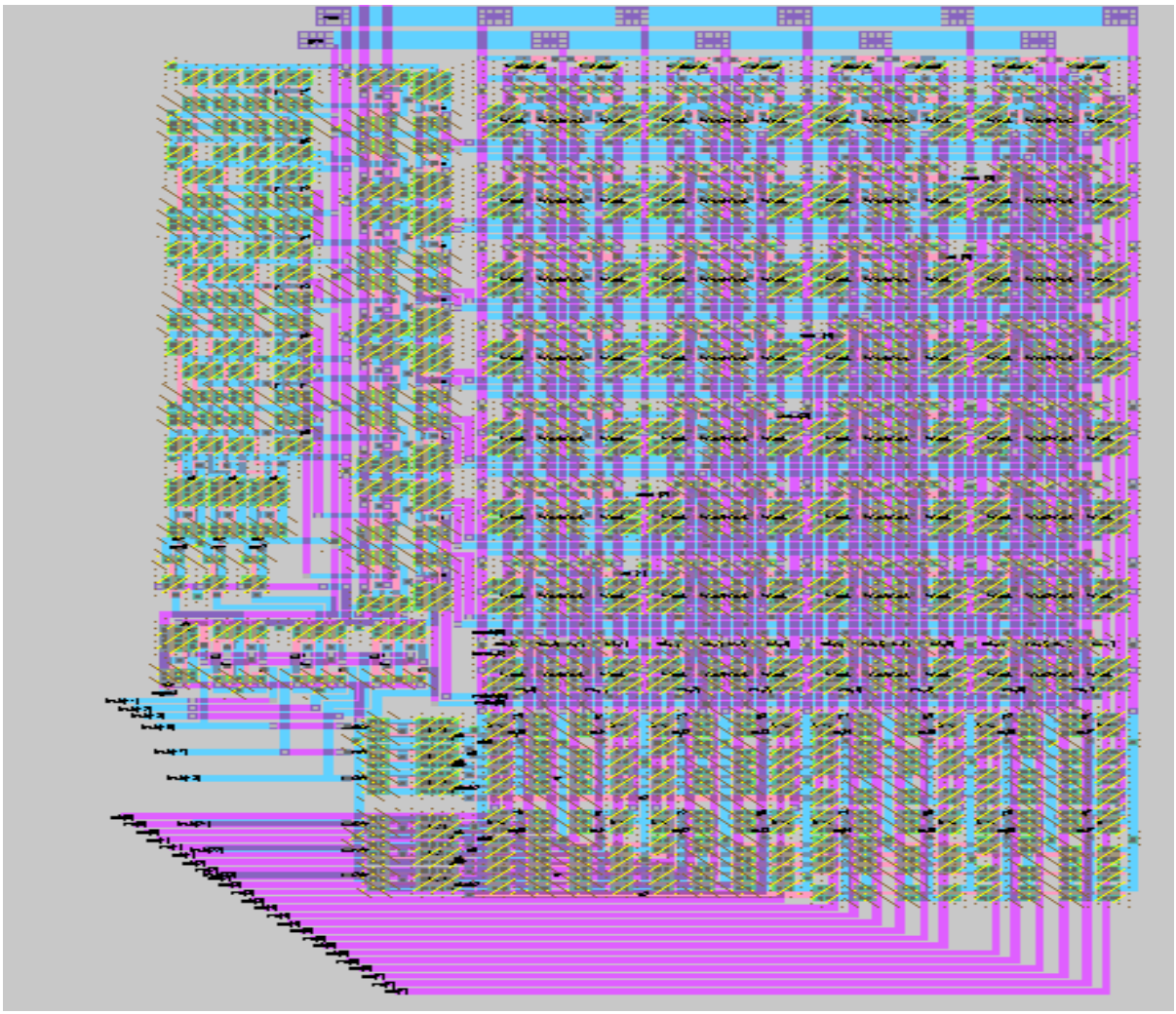
Weak pmos Driver

Lines up along the Register slice and drives a weak one to the registers. The registers have stronger pull-down nmos transistors to overcome this.



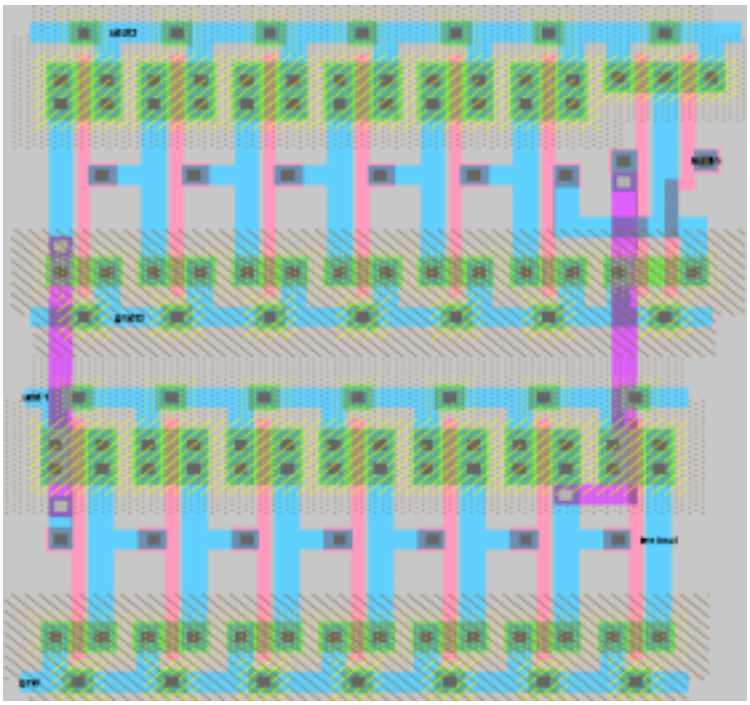
Register Array

Control signals generally come in from the top although there is room to wire in the control signals along metal 2 horizontally. Each byte is a vertical slice of eight bits and there are eight bytes. This is a total of eight 8 bit registers. The weak pmos drivers are located on the right to provide a high. These registers are custom built and mirrored to save space.



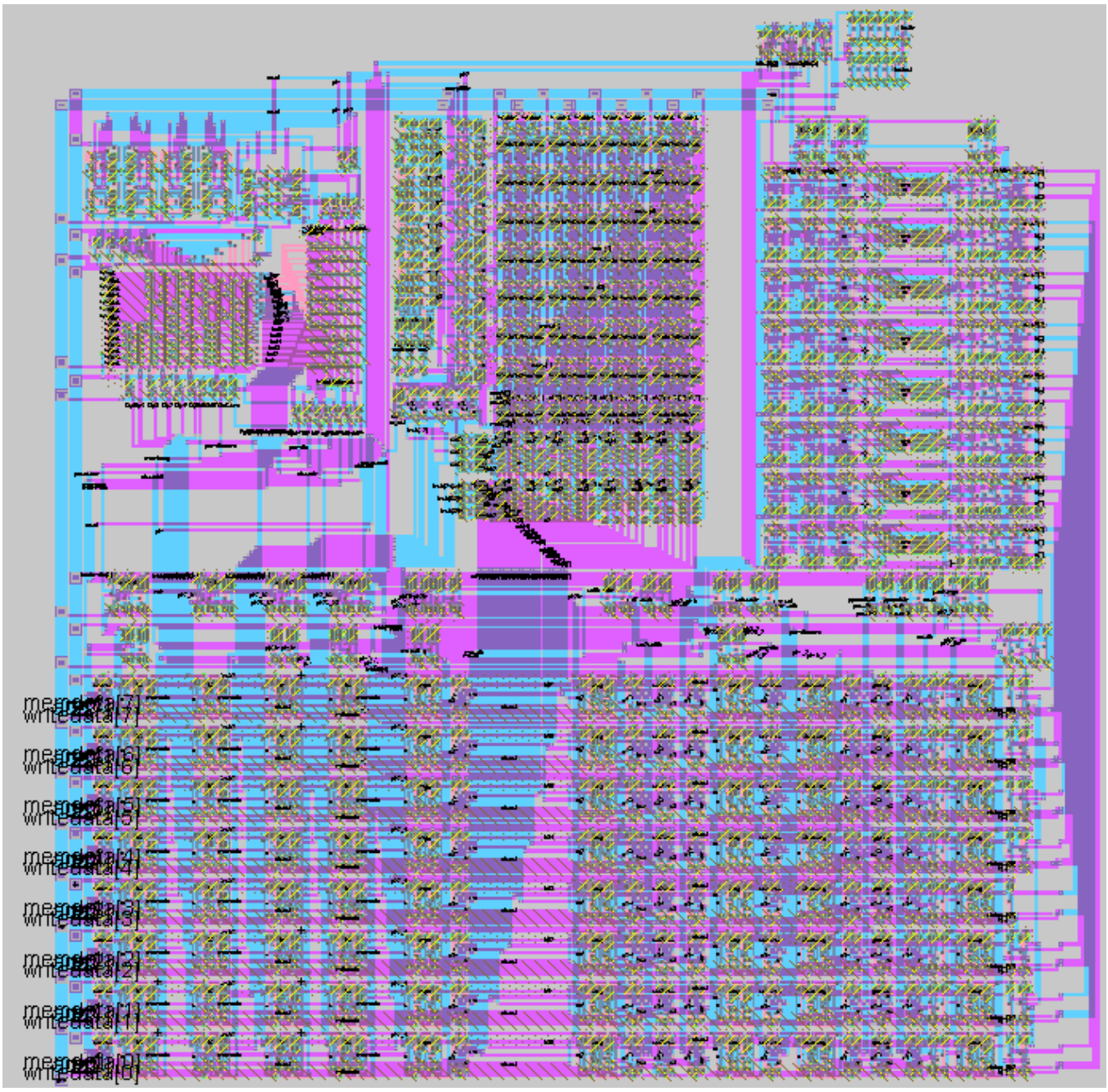
General Registers

The register array is composed of 8 register slices for a total of eight 8-bit registers. There are weak pmos drivers at the end. One decoder for the write signal is on the left hand side and these signals are buffered for true and complementary true and sent to the register array. Control signals for read A and read B are generated in the decoders on the bottom. The decoders are placed here to save space in the horizontal direction. Note that the control signals for the reads are actually ran vertically in metal 2 (the register slice is on its side), this was done to save space and because there was enough room in the registers to run more control with metal 2. The register destination selection mux is included on the bottom of the decoder because there was enough space with an input from the controller on the left. Register Write control signal from the controller is connected to a series of nand 2 gates for the write signals that only allows write to be true when Reg Write is enabled. Notice some of the extraordinarily wide I/O bus to the datapath ran in metal 2 (the metal 2 wires on the bottom of the cell) can overlap with the decoders since there is no metal 2 in the decoders. This saves more than 100 λ in the vertical direction. Instructions for addresses are ran in metal 1 to allow overlap. This passes all checks.



Oscillator

This is implemented on a standard 60 λ pitch with metal 1 running horizontally and metal 2 running vertically. This is a simple circuit used only for testing, so space and design time were minimized. The input and output are connected to external I/O pads. This passes all checks.



8-bit MIPS Microprocessor

Top level layout of all modules linked together properly. The datapath occupies the entire bottom half. Controller implemented as a PLA is in the upper left with many control signals coming out of it. Most control signals are routed down to the datapath, mostly through the zipper. Two signals are sent just to the General Registers which are in the upper middle section. ALU op signals are sent above the registers to the ALU Control which is in the upper right corner which sends buffered signals down to the 8 bit ALU on the upper right side. There is no standard metal 1, metal2 direction since some modules are more efficiently implemented in one direction than another. Interface between modules often shows bending metal 1 or 2. However, this is also used extremely advantageously in bus line to from the datapath to the ALU where metal 1 is

layed directly over metal 2. Read data 1, 2, and write data to and from the registers goes from the center of the datapath to the registers. External signals from the datapath are clumped on the left hand side and will have to be distributed to reach I/O pads. The ring oscillator is in the top left corner. This top level cell passes DRC, ERC, and all NCC checks.