Zack Smaridge
Assignment 4
3-10-12

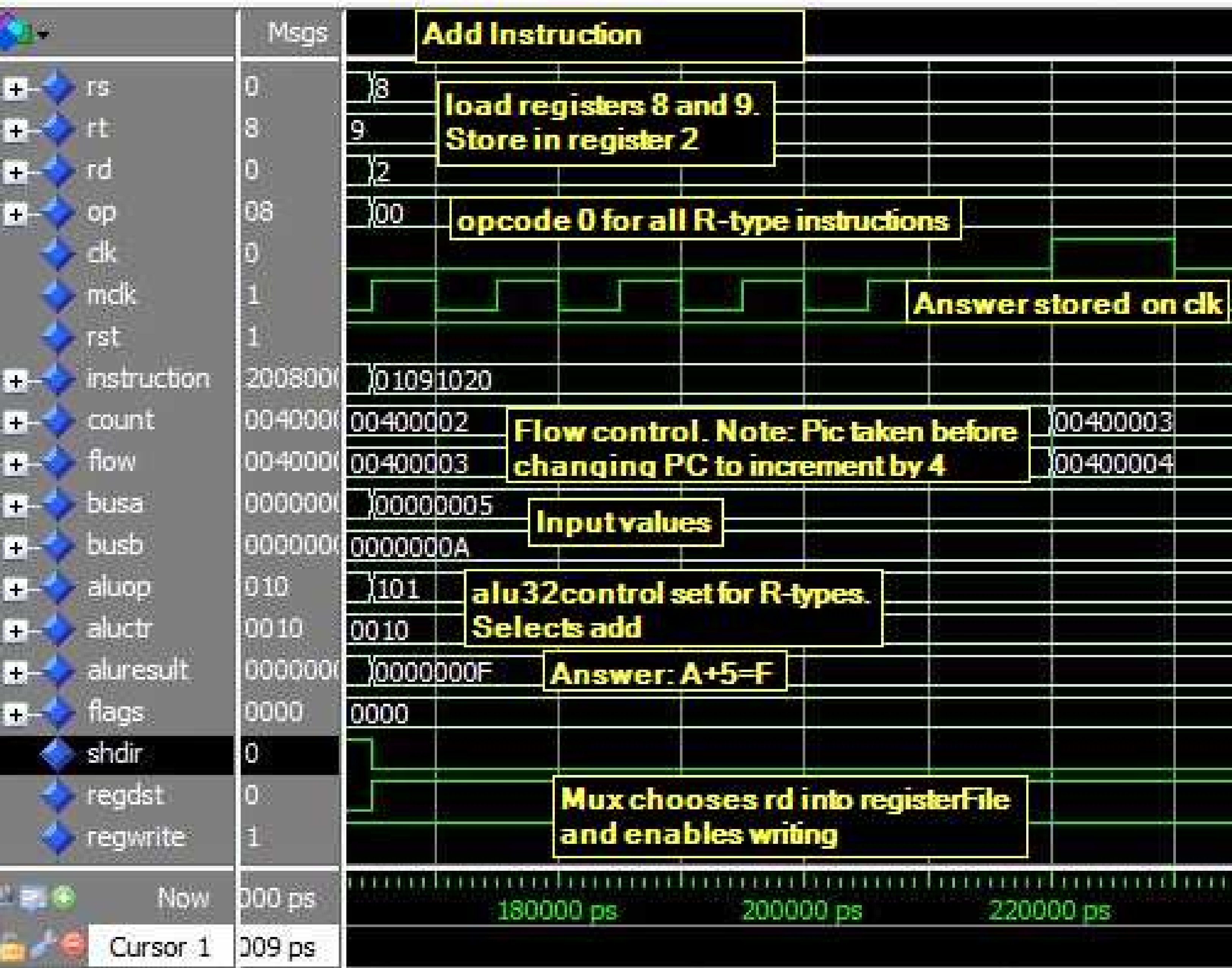
The MIPS processor has 29 core instructions that we implemented. For simplicity we will start with the basic R-type instructions and then move on to the more complicated I and J type instructions. However, please note that for testing I implemented the addi instruction to initialize the registers. This is an initialization that occurs before all the ALU operation instructions.

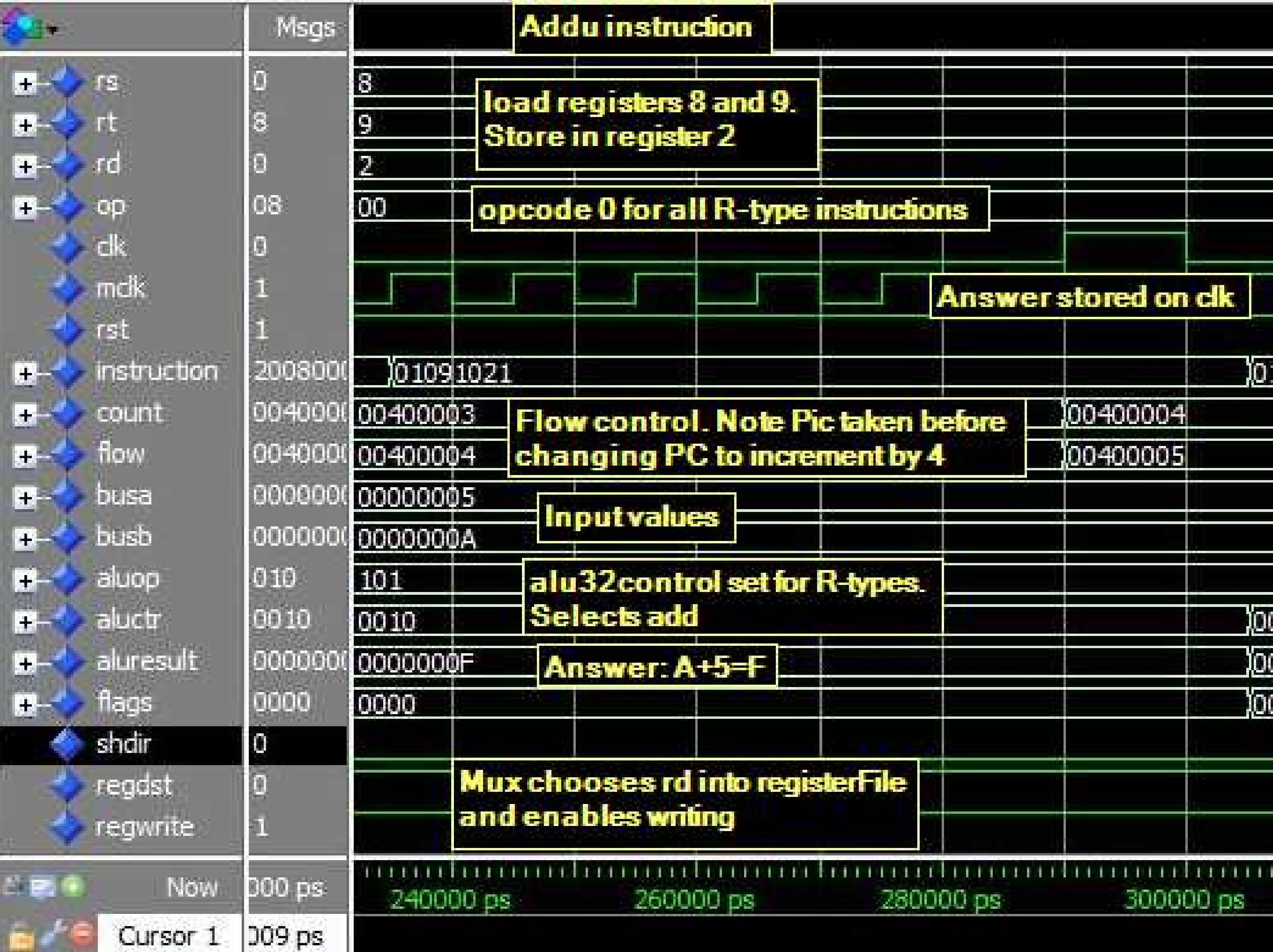
Basic R-type instructions:
Covers: add, addu, and, nor, or, slt, sltu, sll, srl, sub, subu
Not included: jr
Hardware: PC, IM, Control, registerFile, alu32control, ALU, add32
Description: The basic hardware needed to execute these instructions starts with a program counter that is essentially a 30-bit register. This feeds into the address line of the instruction memory created from a 32-bit asynchronous RAM. The 2 LSB bits are made to be zero's to account for the two missing bits. The next piece of basic hardware is a 32 by 32-bit register file. The 32-bit instruction is fed into the two read ports and write port using bits 25 to 11. The two read outputs of the register file are then fed into the two inputs of the 32-bit ALU. The output of the ALU wraps back around to be input into the write data of the register file. We also need to include our alu32control module to provide the ALU with the proper control signals based on the ALUop from the controller (to be covered) and the last 6 bits of the instruction representing the function field in R-type instructions. We also need to implement a 32-bit adder to increment our program counter such that we can step through instructions. The inputs to the adder will be the previous count and hardwired to 4. The last and most vital piece of hardware for all of these instructions is the controller. The controller is responsible for 2 signals at this point: regWrite and ALUop. The input to the controller is instruction[31-26] which represents the opcode. At this point the controller is very simple and by default sets the ALUop for R-type instructions and the register file is set to write a new value.

The table below illustrates the R-type.mif used to simulate the 11 separate waveforms on the following pages in the order listed under "hardware."

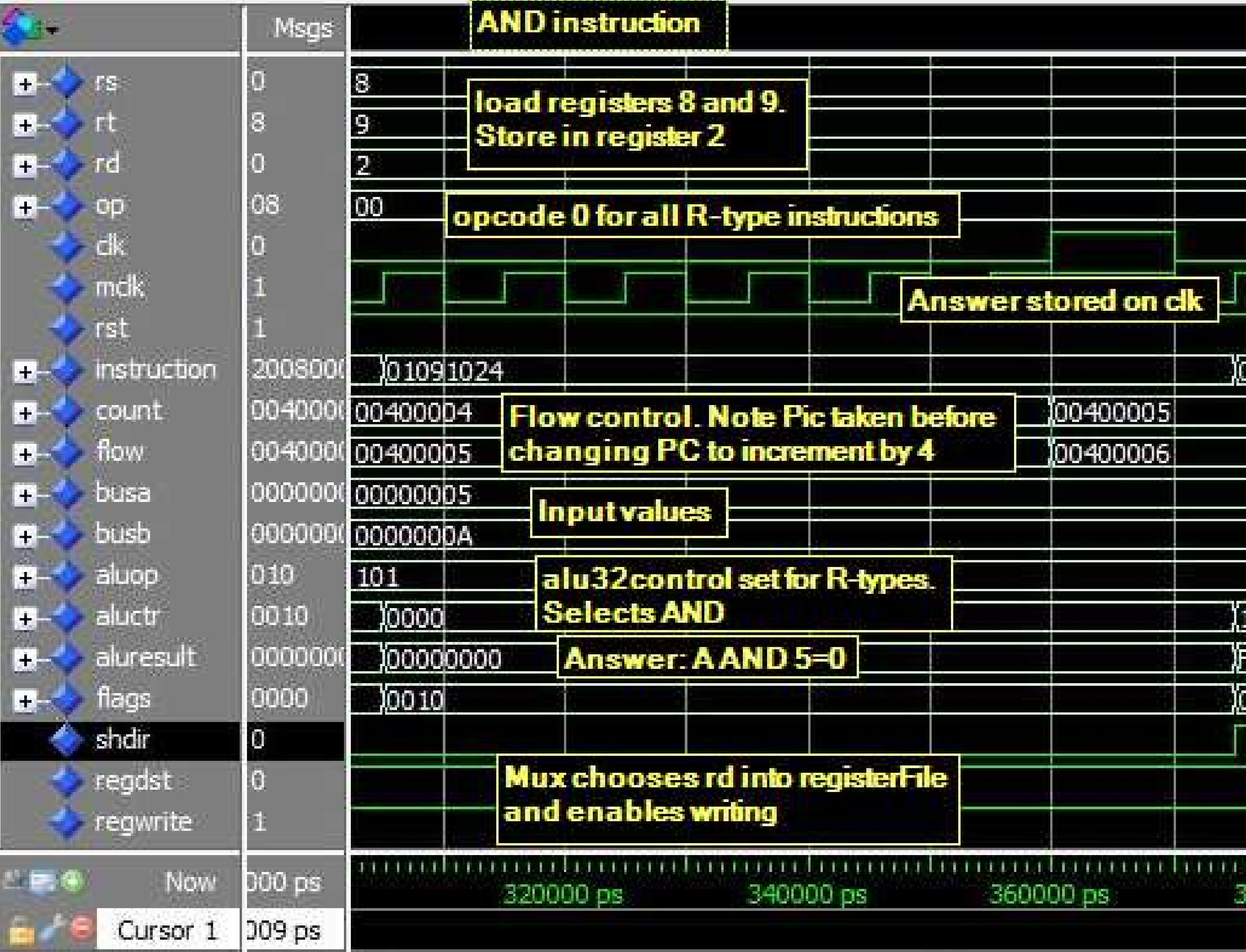
Instruction	Opcode rs rt rd shamt function
addi	0010.00 00.000 0.1000. 0000.0 000.00 00.0101
addi	0010.00 00.000 0.1001. 0000.0 000.00 00.0101
add	0000.00 01.000 0.1001. 0001.0 000.00 10.0000
	$v0 = $t0 + $t1
addu	000000 00010 01000 01001 00000 100001
	$v0 = $t0 + $t1
AND	000000 00010 01000 01001 00000 100100
	$v0 = $t0 AND $t1
NOR	000000 00010 01000 01001 00000 100111
	$v0 = $t0 NOR $t1
OR	000000 00010 01000 01001 00000 100101
	$v0 = $t0 OR $t1
slt	000000 00010 01000 01001 00000 101010
	$v0 = 1 if($t0 < $t1)
sltu	000000 00010 01000 01001 00000 101011
	$v0 = 1 if($t0 < $t1)
sll	000000 00010 01000 01001 00010 000000
	$v0 = $t1 <-
srl	000000 00010 01000 01001 00010 000010
	$v0 = $t1 ->

```
sub     000000 00010 01000 01001 00000 100010
                $v0  =  $t0 -  $t1
subu    000000 00010 01000 01001 00000 100011
                $v0  =  $t0 -  $t1
jr      000000 00010 00000 00000 00000 001000
```

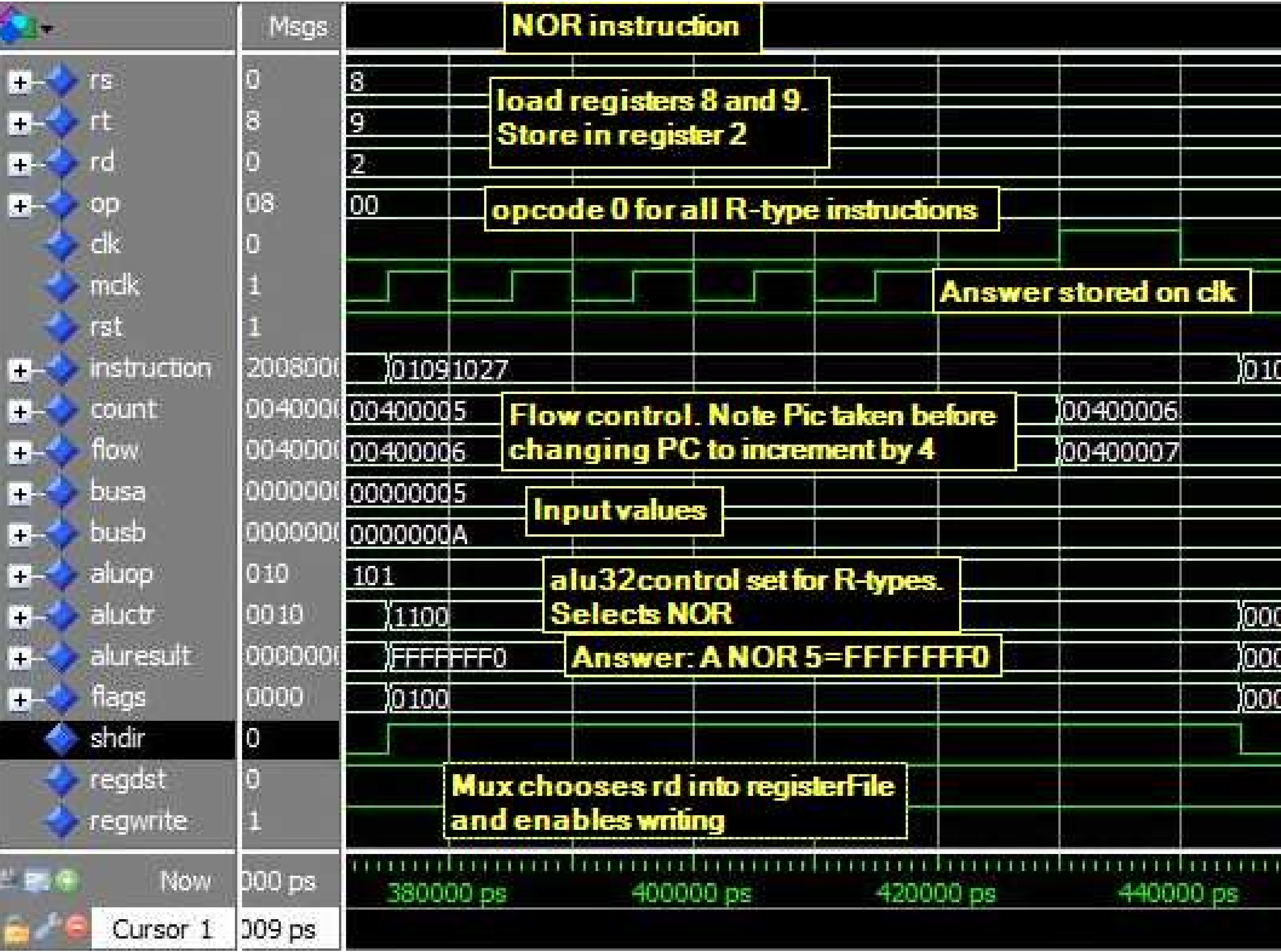

Msgs
Add Instruction
rs
rt
rd
op
clk
mclk
rst
instruction
count
flow
busa
busb
aluop
aluctr
aluresult
flags
shdir
regdst
regwrite
load registers 8 and 9.
Store in register 2
opcode 0 for all R-type instructions
Answer stored on clk
01091020
00400002
00400003
Flow control. Note: Pic taken before
changing PC to increment by 4
00400003
00400004
00000005
Input values
0000000A
101
alu32control set for R-types.
Selects add
0010
0000000F
Answer: A+5=F
0000
Mux chooses rd into registerFile
and enables writing
Now
Cursor 1
180000 ps
200000 ps
220000 ps



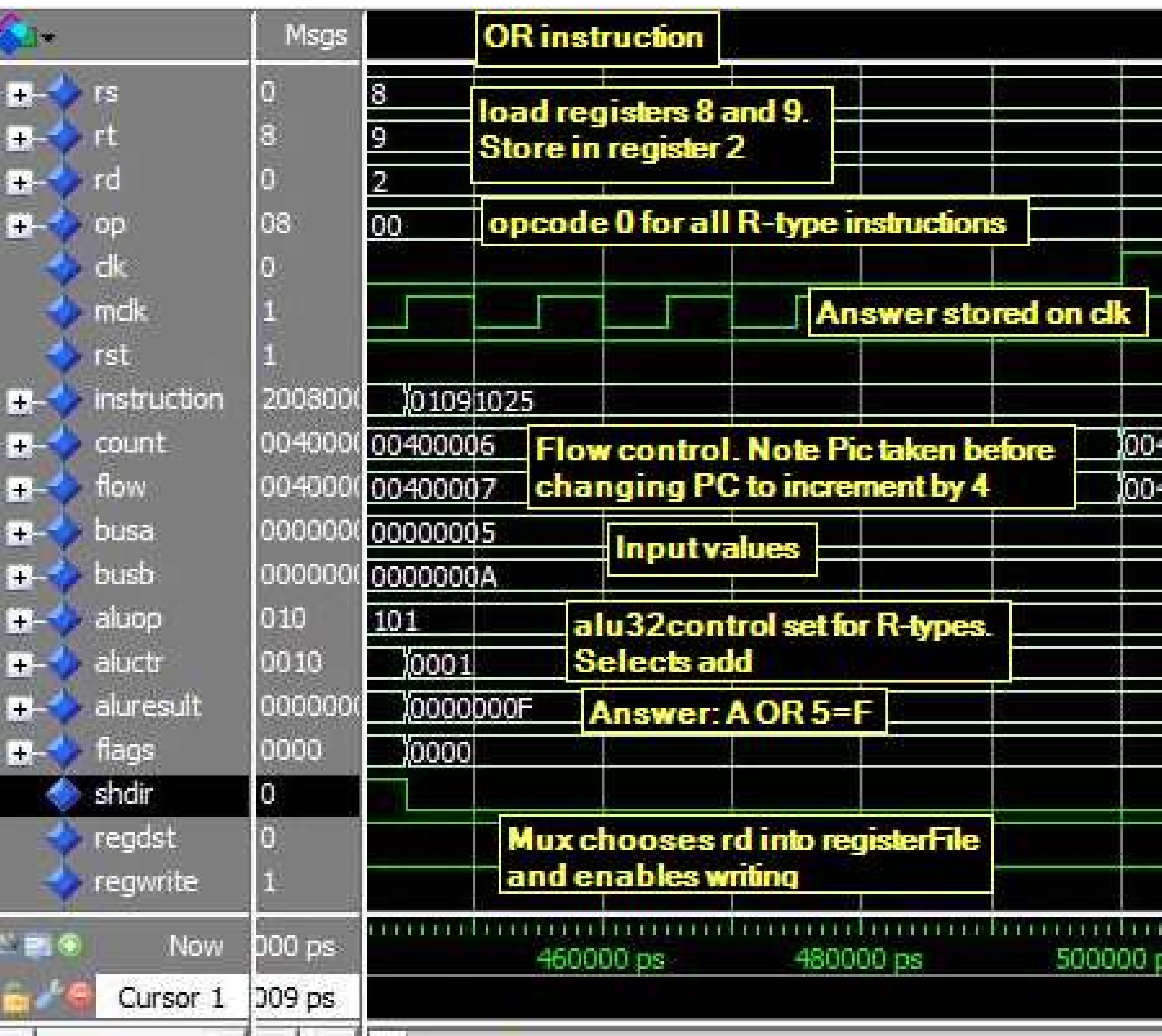
Msgs
Addu instruction
rs
rt
rd
op
clk
mclk
rst
instruction
count
flow
busa
busb
aluop
aluctr
aluresult
flags
shdir
regdst
regwrite
load registers 8 and 9.
Store in register 2
opcode 0 for all R-type instructions
Answer stored on clk
01091021
00400003
00400004
Flow control. Note Pic taken before
changing PC to increment by 4
00400004
00400005
00000005
0000000A
Input values
101
0010
alu32control set for R-types.
Selects add
0000000F
Answer: A+5=F
0000
Mux chooses rd into registerFile
and enables writing
Now
Cursor 1
240000 ps
260000 ps
280000 ps
300000 ps



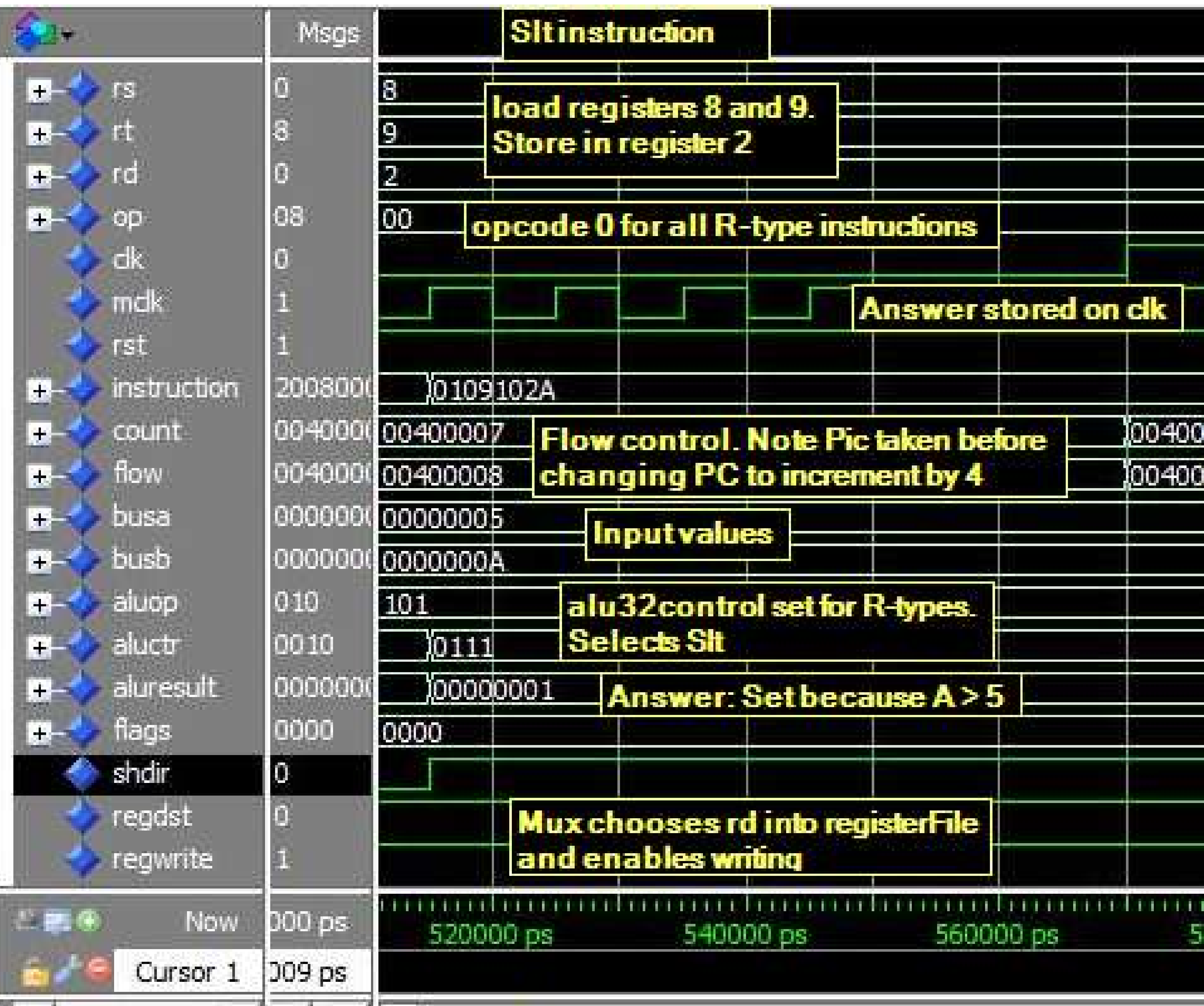
AND instruction
Msgs
rs
rt
rd
op
clk
mclk
rst
instruction
count
flow
busa
busb
aluop
aluctr
aluresult
flags
shdir
regdst
regwrite
load registers 8 and 9.
Store in register 2
opcode 0 for all R-type instructions
Answer stored on clk
Flow control. Note Pic taken before
changing PC to increment by 4
Input values
alu32control set for R-types.
Selects AND
Answer: A AND 5=0
Mux chooses rd into registerFile
and enables writing
01091024
00400004
00400005
00400006
00000005
0000000A
101
0000
00000000
0010
Now
Cursor 1
320000 ps
340000 ps
360000 ps

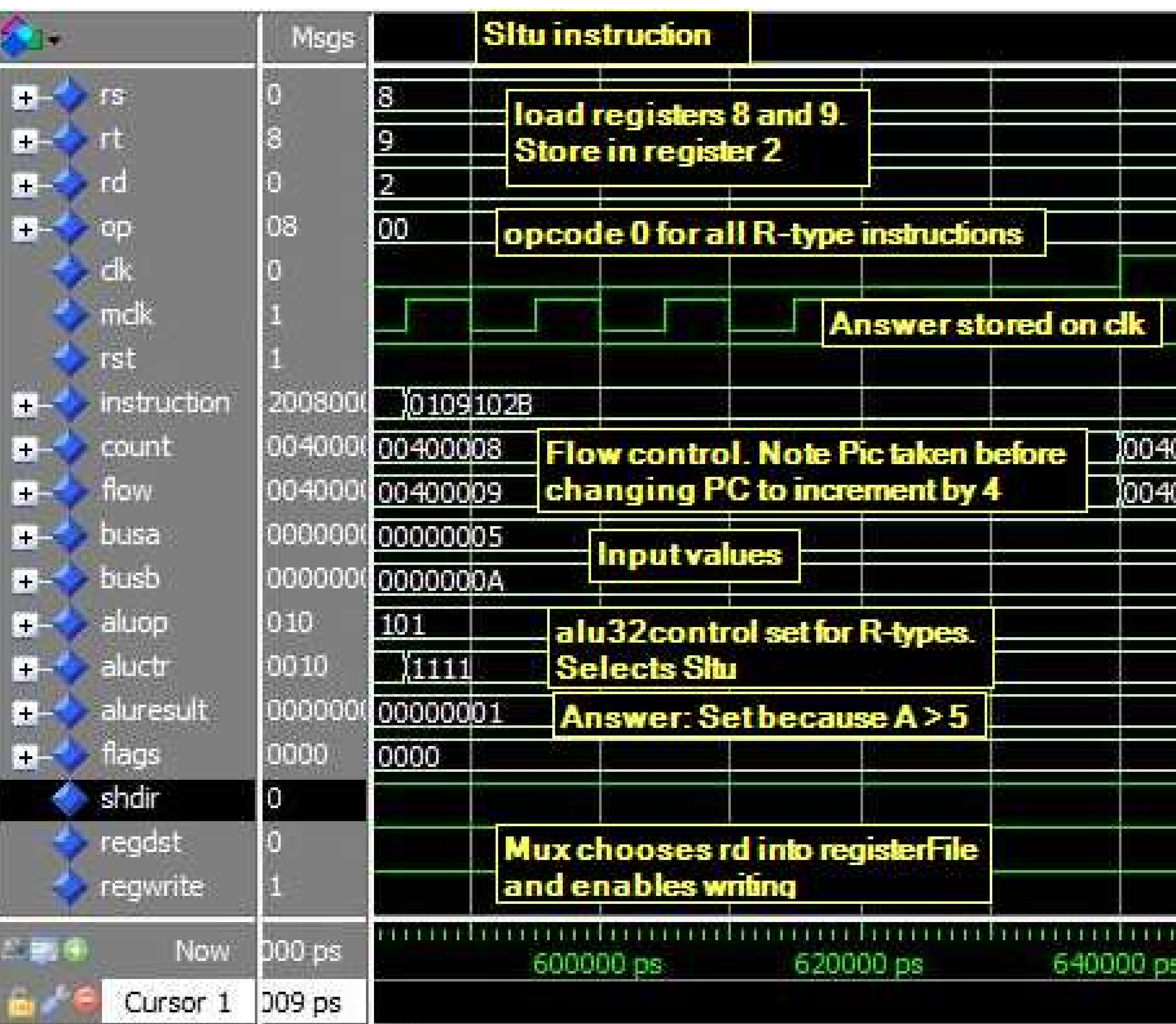


Msgs
NOR instruction
rs
rt
rd
op
clk
mclk
rst
instruction
count
flow
busa
busb
aluop
aluctr
aluresult
flags
shdir
regdst
regwrite
load registers 8 and 9.
Store in register 2
opcode 0 for all R-type instructions
Answer stored on clk
01091027
00400005
00400006
00400007
00000005
0000000A
101
1100
FFFFFFF0
0100
Flow control. Note Pic taken before changing PC to increment by 4
Input values
alu32control set for R-types.
Selects NOR
Answer: A NOR 5=FFFFFFF0
Mux chooses rd into registerFile and enables writing
Now
Cursor 1
380000 ps
400000 ps
420000 ps
440000 ps

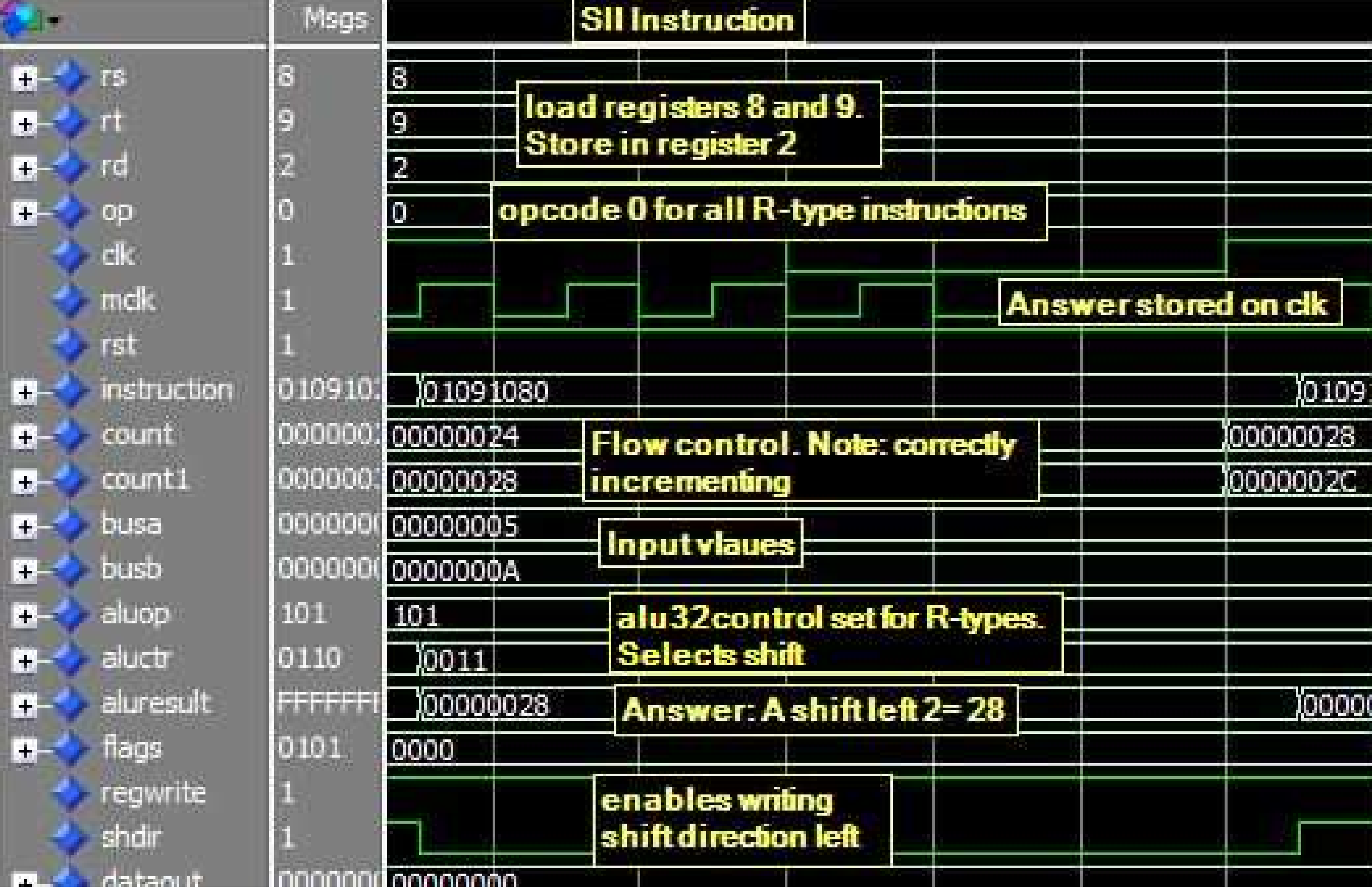


OR instruction
load registers 8 and 9.
Store in register 2
opcode 0 for all R-type instructions
Answer stored on clk
Flow control. Note Pic taken before changing PC to increment by 4
Input values
alu32control set for R-types.
Selects add
Answer: A OR 5=F
Mux chooses rd into registerFile and enables writing
rs
rt
rd
op
clk
mclk
rst
instruction
count
flow
busa
busb
aluop
aluctr
aluresult
flags
shdir
regdst
regwrite
Msgs
Now
Cursor 1
01091025
00400006
00400007
00000005
0000000A
101
0001
0000000F
0000
460000 ps
480000 ps


Slt instruction
load registers 8 and 9.
Store in register 2
opcode 0 for all R-type instructions
Answer stored on clk
Flow control. Note Pic taken before
changing PC to increment by 4
Input values
alu32control set for R-types.
Selects Slt
Answer: Set because A > 5
Mux chooses rd into registerFile
and enables writing
rs
rt
rd
op
clk
mclk
rst
instruction
count
flow
busa
busb
aluop
aluctr
aluresult
flags
shdir
regdst
regwrite
Msgs
Now
Cursor 1
520000 ps
540000 ps
560000 ps



Msgs
Sltu instruction
rs
rt
rd
op
clk
mclk
rst
instruction
count
flow
busa
busb
aluop
aluctr
aluresult
flags
shdir
regdst
regwrite
load registers 8 and 9.
Store in register 2
opcode 0 for all R-type instructions
Answer stored on clk
0109102B
00400008
00400009
Flow control. Note Pic taken before changing PC to increment by 4
00000005
0000000A
Input values
101
1111
alu32control set for R-types.
Selects Sltu
00000001
Answer: Set because A > 5
0000
Mux chooses rd into registerFile and enables writing
Now
Cursor 1
600000 ps
620000 ps



Msgs
Sll Instruction
rs
rt
rd
op
clk
mclk
rst
instruction
count
count1
busa
busb
aluop
aluctr
aluresult
flags
regwrite
shdir
dataout
load registers 8 and 9.
Store in register 2
opcode 0 for all R-type instructions
Answer stored on clk
01091080
00000024
00000028
00000028
0000002C
Flow control. Note: correctly incrementing
00000005
0000000A
Input vlaues
101
0011
0110
alu32control set for R-types.
Selects shift
00000028
Answer: A shift left 2= 28
0000
0101
enables writing
shift direction left
00000000

Signal	Msgs	Waveform
rs	8	8
rt	9	9
rd	2	2
op	0	0
clk	0	
mclk	1	
rst	1	
instruction	0109101	01091082 … 0109
count	0000003	00000028 … 0000002C
count1	0000003	0000002C … 00000030
busa	0000000	00000005
busb	0000000	0000000A
aluop	101	101
aluctr	0110	0011 … 0110
aluresult	FFFFFFF	00000002 … FFFF
flags	0101	0000 … 0101
regwrite	1	
shdir	1	
dataout	0000000	00000000

Srl Instruction

load registers 8 and 9.
Store in register 2

opcode 0 for all R-type instructions

Answer stored on clk

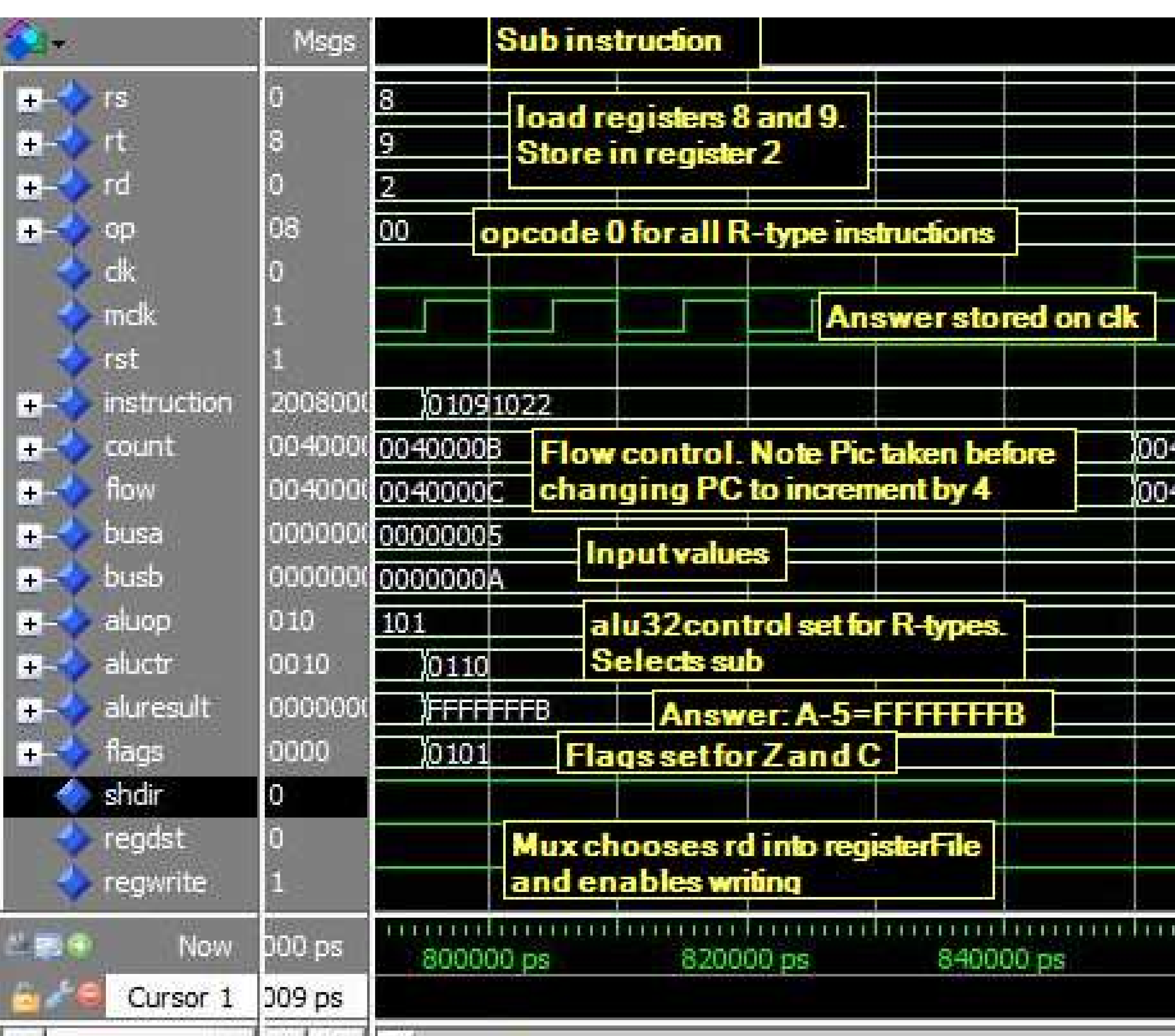
Flow control. Note:
correct incrementation

Input value

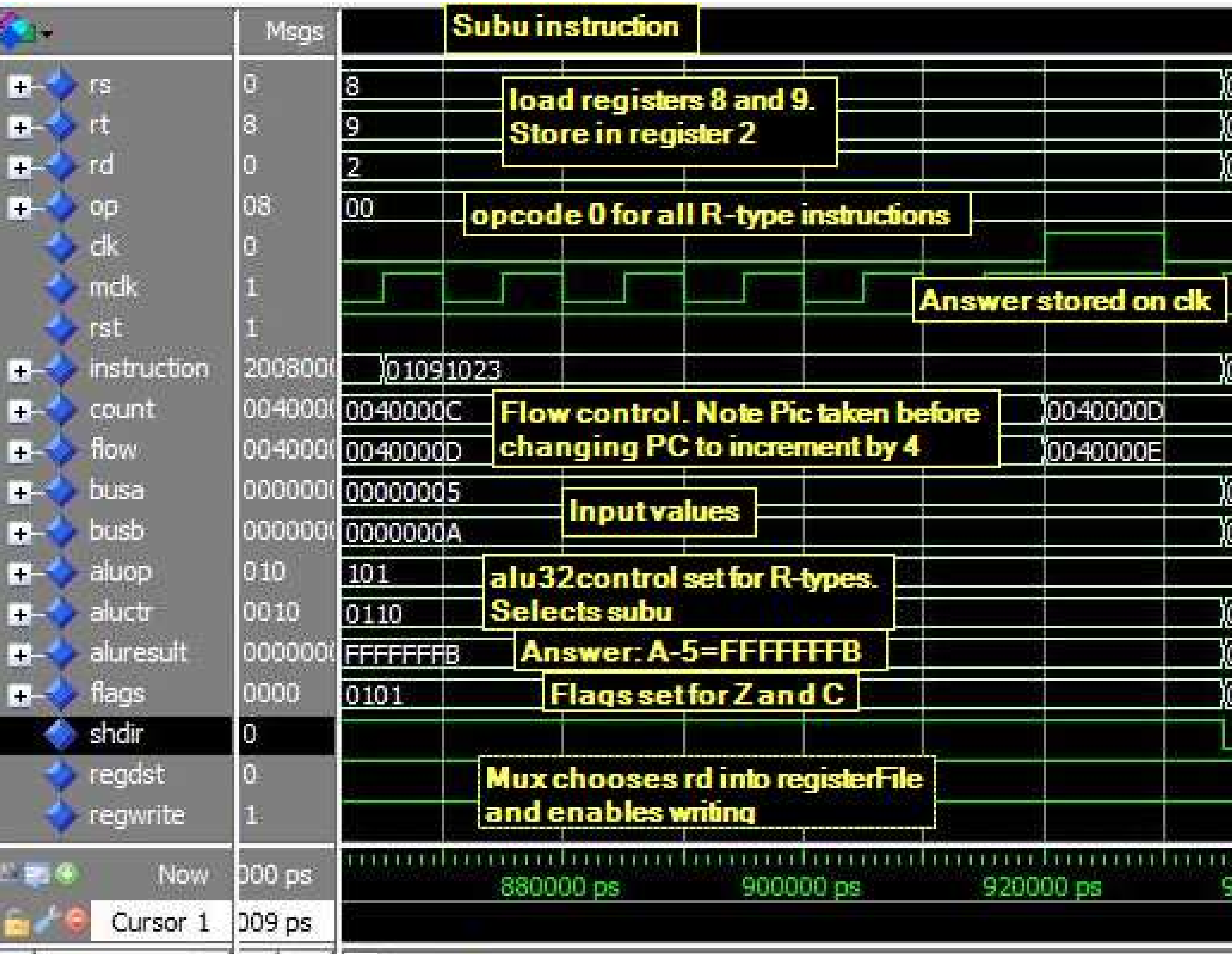
alu32control set for R-types.
Selects shift

Answer: A shift right 2 = 2

Enable writing to registerFile.
Shift set to right



Sub instruction
load registers 8 and 9.
Store in register 2
opcode 0 for all R-type instructions
Answer stored on clk
Flow control. Note Pic taken before changing PC to increment by 4
Input values
alu32control set for R-types. Selects sub
Answer: A-5=FFFFFFFB
Flags set for Z and C
Mux chooses rd into registerFile and enables writing



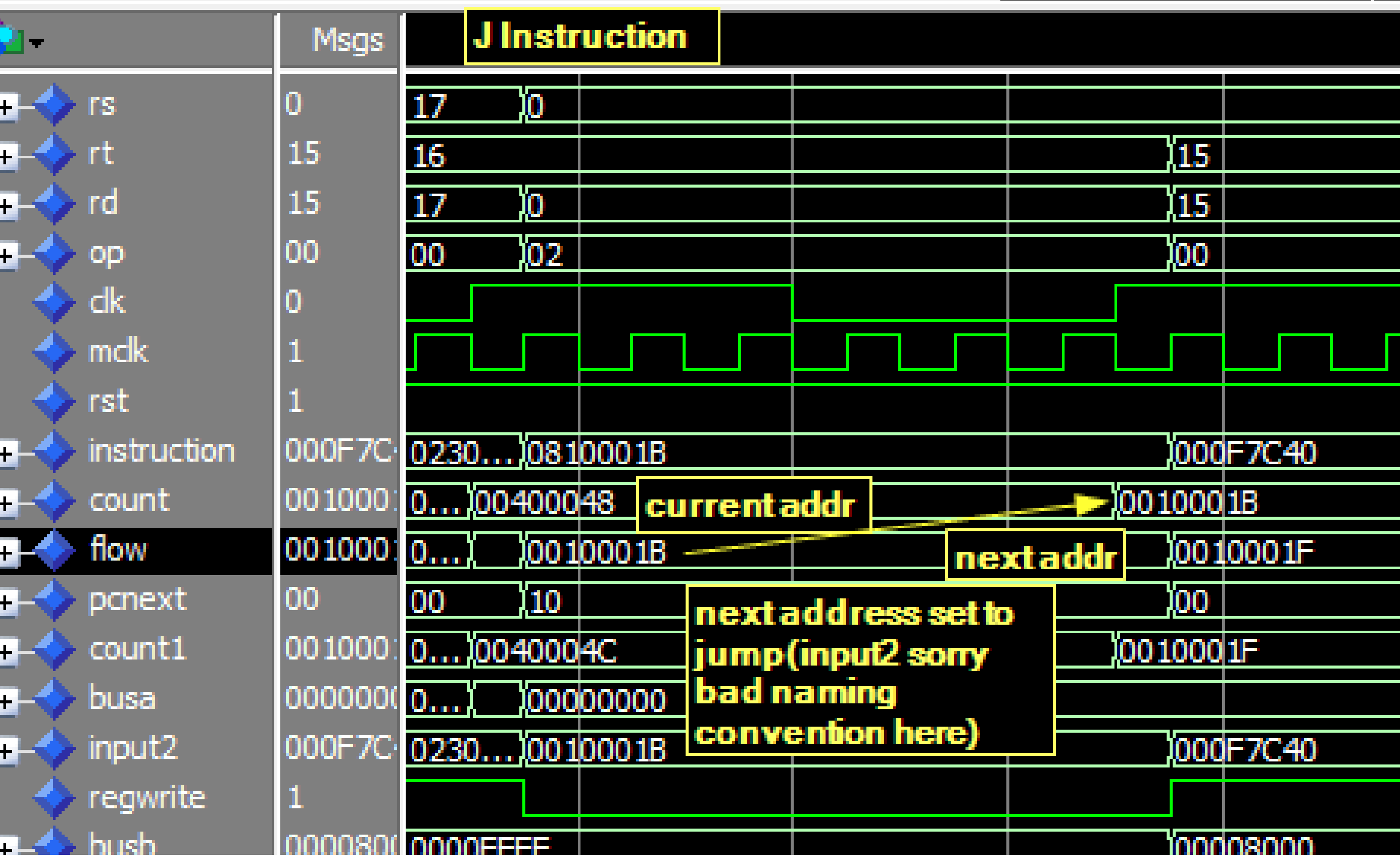
Msgs
Subu instruction
rs
rt
rd
op
clk
mclk
rst
instruction
count
flow
busa
busb
aluop
aluctr
aluresult
flags
shdir
regdst
regwrite
load registers 8 and 9.
Store in register 2
opcode 0 for all R-type instructions
Answer stored on clk
01091023
0040000C
0040000D
0040000E
Flow control. Note Pic taken before
changing PC to increment by 4
00000005
0000000A
Input values
alu32control set for R-types.
Selects subu
FFFFFFFB
Answer: A-5=FFFFFFFB
0101
Flags set for Z and C
Mux chooses rd into registerFile
and enables writing
Now
Cursor 1
880000 ps
900000 ps
920000 ps

R-type Jump Instruction:

Added Hardware: 32-bit control flow mux

Description: To implement the jump register instruction we simply add a mux going into the program counter that can decide between the output of busA or the normal incrementation. We then add the control signal PCnext for the mux select line. The final step is to add the logic to change PCnext depending on the function field. At this point the controller still only has one case for all the R-type instructions and the defaults are all set for it.

The next page is the wave diagram for the jump register instruction.


J Instruction
Msgs
rs
rt
rd
op
clk
mclk
rst
instruction
count
flow
pcnext
count1
busa
input2
regwrite
busb
current addr
next addr
next address set to jump(input2 sorry bad naming convention here)
08100001B
00400048
0010001B
0010001F
0040004C
00000000
000F7C40
0000FFFF
00008000

Basic I-Type Instructions:

Covers: addi, addiu, andi, ori, slti, sltiu

Not Included: beg, bne, lbu, lhu, lui, lw, sb, sh, sw

Hardware: Mux going into ALU source, Mux going into rw of registerFile, SignExtender

Description: Implementing the basic arithmetic I-type instructions requires three new pieces of hardware. The first two are muxs that allow an immediate value to be input into the ALU port B and rt to go into the destination register address. The third addition is the SignExtender which takes in the last 16 bits of the instruction and extends it into 32-bits for the ALU to use. The control logic added is a select line for each mux: ALUsrc and RegDest. The SignExtender also needs a control signal to choose between zero and sign extension. The controller also requires a case statement for each immediate instruction we have created dictated by the opcode. The default controls set previously for the R-type instructions must be adjusted in each case statement so that the registerFile write enable is not set.

The table below illustrates the I-types.mif used to simulate the waveforms on the following pages.

Instruction	Opcode	rs	rt	immidiate
addi	0010.00	00.000	0.1000.	0000.0000.0000.0101
		r8=r0+0x5=0x5		
addiu	0010.01	01.000	0.1001.	1111.0000.0000.1010
		r9=r8+0x0000F00A=0x0000F00F		
andi	0011.00	01.001	0.1010.	0000.0000.0011.1100
		rA=r9 AND 0x3C=0xC		
ori	0011.01	01.000	0.1001	0000.0000.0000.1010
		r9=r8 OR A=0xF		
slti	0010.10	01.001	0.1000	1111.0000.0000.0000
		r8=0, r9>F		
sltiu	0010.11	01.001	0.1000	1111.0000.0000.0000
		r8=1, r9<F		

Addi Instruction

Signal	Msgs	Waveform values
rs	16	0, 16
rt	9	0, 8, 9
rd	30	0, 30
op	09	00, 08, 09
clk	0	
mclk	0	
rst	1	
instruction	2609F00	0000..., 20080005, 26
count	0040000	00400000, 00400004
count1	0040000	00400004, 00400008
busa	0000000	00000000
busb	0000F00	0000..., 00000005, 00
immediate	0000F00	0000..., 00000005, 00
imdsign	0	
alusrc	1	
aluop	010	101, 010
aluctr	0010	0011, 0010
aluresult	0000F00	XXXX..., 00000005, 00
regdst	00	01, 00
regwrite	1	
dataout	0000000	00000000

load register 0.
Store n register 8

opcode for add

Answer stored on clk

Flow control. Note:
correct incrementation

Immidiate value
loaded on busB.
Sign extension

I-type. Select add

Answer: 0+5=5

registerFile destination set to rt.
Write enabled

Addiu Instruction

Signal	Msgs	Waveform values
rs	9	0, 8, 9
rt	10	8, 9, 10
rd	0	0, 30, 0
op	0C	08, 09, 0C
clk	1	
mclk	0	
rst	1	
instruction	312A00	200…, 2509F00A, 312A00
count	0040000	00400004, 00400008
count1	0040000	00400008, 0040000C
busa	0000F0	000…, 00000005, 0000F0
busb	000000	000…, 0000F00A, 000000
immediate	000000	000…, 0000F00A, 000000
imdsign	0	
alusrc	1	
aluop	000	010, 000
aluctr	0000	0010, 0000
aluresult	000000	000…, 0000F00F, 000000
regdst	00	00
regwrite	1	
dataout	000000	00000000

load register 8.
Store n register 9

opcode for addu

Answer stored on clk

Flow control. Note:
correct incrementation

Immidiate value
loaded on busB.
Unsigned extension

I-type. Select add

Answer: 5+F00A=F00F

registerFile destination set to rt
Write enabled

Andi Instruction

Signal	Msgs	Values
rs	8	8 9 8
rt	9	9 10 9
rd	0	30 0
op	0D	09 0C 0D
clk	1	
mclk	1	
rst	1	
instruction	350900	25... 312A003C 350900
count	0040000	00400008 0040000C
count1	004000	0040000C 00400010
busa	0000000	00... 0000F00F 000000
busb	0000000	00... 0000003C 000000
immediate	0000000	00... 0000003C 000000
imdsign	0	
alusrc	1	
aluop	110	010 000 110
aluctr	0001	0010 0000 0001
aluresult	0000000	00... 0000000C 000000
regdst	00	00
regwrite	1	
dataout	0000000	00000000

load register 9.
Store n register A

opcode for AND

Answer stored on clk

Flow control. Note:
correct incrementation

Immidiate value
loaded on busB.
Unsigned extension

I-type. Select AND

Answer: F00F AND 3C=C

registerFile destination set to rt.
Write enabled

Ori Instruction

Signal	Msgs	Waveform values
rs	9	9, 8, 9
rt	8	10, 9, 8
rd	30	0, 30
op	0A	0C, 0D, 0A
clk	0	
mclk	0	
rst	1	
instruction	2928F0	312..., 3509000A, 2928F0
count	004000	0040000C, 00400010
count1	004000	00400010, 00400014
busa	0000000	0000..., 00000005, 000000
busb	FFFFF0	0000..., 0000000A, FFFFF0
immediate	FFFFF0	0000..., 0000000A, FFFFF0
imdsign	1	
alusrc	1	
aluop	111	000, 110, 111
aluctr	0111	0000, 0001, 0111
aluresult	0000000	0000..., 0000000F, 000000
regdst	00	00
regwrite	1	
dataout	000000	00000000

load register 8.
Store n register 9

opcode for OR

Answer stored on clk

Flow control. Note:
correct incrementation

Immidiate value
loaded on busB.
Unsigned extension

I-type. Select OR

Answer: 5 OR A = F

registerFile destination set to rt.
Write enabled

Slti Instruction

Signal	Msgs	Values
rs	9	8, 9
rt	8	9, 8
rd	30	0, 30
op	0B	0D, 0A, 0B
clk	0	
mclk	0	
rst	1	
instruction	2D28F0	35090…, 2928F000, 2D28F0
count	004000	0…, 00400010, 00400014
count1	004000	0…, 00400014, 00400018
busa	000000	00000…, 0000000F
busb	0000F0	00000…, FFFFF000, 0000F0
immediate	0000F0	00000…, FFFFF000, 0000F0
imdsign	0	
alusrc	1	
aluop	100	110, 111, 100
aluctr	1111	0001, 0111, 1111
aluresult	000000	00000…, 00000000, 00000
regdst	00	00
regwrite	1	
dataout	000000	00000000

load register 9. Store in register 8

opcode for slt

Answer stored on clk

Flow control. Note: correct incrementation

Immidiate value loaded on busB. Sign extension

I-type. Select slt

Answer: 0; 0000000F>FFFFF000

registerFile destination set to rt. Write enabled

Sltiu Instruction

Signal	Msgs	Values
rs	0	9 → 0
rt	0	8 → 0
rd	0	30 → 0
op	00	0B → 00
clk	1	
mclk	0	
rst	1	
instruction	0000000(	2D28F000 → 00000
count	004000	00400014 → 00400018
count1	004000	00400018 → 0040001C
busa	0000000(	0000000F → 00000
busb	0000000(	0000F000 → 00000
immediate	0000000(	0000F000 → 00000
imdsign	1	
alusrc	0	
aluop	101	100 → 101
aluctr	0011	1111 → 0011
aluresult	0000000(	00000001 → 00000
regdst	01	00 → 01
regwrite	1	
dataout	0000000(	00000000

load register 9. Store in register 8

opcode for sltu

Answer stored on clk

Flow control. Note: correct incrementation

Immidiate value loaded on busB. Unsigned extension

I-type. Select sltu

Answer: 1; 0000000F<0000F000

registerFile destination set to rt. Write enabled

I-type Memory Instructions
Covers: lbu, lhu, lui, lw, sb, sh, sw
Not Included: beq, bne only I-types ledt to cover
Hardware: Multiple muxs into write port of registerFile, DataMemory
Description: The main addition for these instructions is the RAM module containing the data memory. The address input will be connected to the result of the ALU and the data input will come from the second output of the register file. In addition, there are 4 control signals that are needed: read enable, write enable and 2 byte enables. The output data is then connected to a series of muxes going into the write port of the registerFile. The first mux is a 4 to 1 that has three separate mask options for word, half word and byte. This mux requires a two bit select line called mask. The output then goes into another 4 to 1 mux before going to the write port of the registerFile. The other inputs to the mux are the pre existing ALUresult and a special word for the lui instruction. The load upper immediate choice is simply the 16 LSB of the immediate input shifted left to the MSB. This new mux also needs a 2-bit control line, MemtoReg. In addition to these new control signals, the controller will also need a new case for each instruction and modified default controls as before.

The table below illustrates the lw_sw.mif used to simulate the waveforms on the following pages. The 7 waveforms are in the order listed above.

Instruction	Opcode	rs	rt	immediate
addi	0010.00	00.000	0.1000.	1111.1111.1111.1111
	r8=r0+0xFFFFFFFF=0xFFFFFFFF			
sw	1010.11	00.000	0.1000	0000.0000.0000.0000
	mem[0(r0)] = r8 = 0xFFFFFFFF			
lw	1000.11	00.000	0.1000	0000.0000.0000.0000
	r8 = mem[0(r0)] = 0xFFFFFFFF			
sh	1010.01	00.000	0.1000	0000.0000.0000.0100
	mem[4(r0)] = r8 = 0x0000FFFF			
lw	1000.11	00.000	0.1000	0000.0000.0000.0100
	r8 = mem[4(r0)] = 0x0000FFFF			
sb	1010.00	00.000	0.1000	0000.0000.0000.1000
	mem[8(r0)] = r8 = 0x000000FF			
lw	1000.11	00.000	0.1000	0000.0000.0000.1000
	r8 = mem[8(r0)] = 0x000000FF			
lhu	1001.01	00.000	0.1000	0000.0000.0000.0000
	r8 = mem[0(r0)] = 0x0000FFFF			
lbu	1001.00	00.000	0.1000	0000.0000.0000.0000
	r8 = mem[0(r0)] = 0x000000FF			
lui	0011.11	00.000	0.1000	1010.1011.1100.1101
	r8 = AB			

Lb Instruction

Signal	Msgs	Waveform values
rs	0	0
rt	8	8
rd	21	0, 21
op	0F	25, 24, 0F
clk	0	
mclk	1	
rst	1	
instruction	3C08AB	90080000, 3C08AB
count	0040002	00400020, 00400024
count1	0040002	00400024, 00400028
busa	0000000	00000000
immediate	FFFFAB	00000000, FFFFAB
aluop	101	010, 101
alusrc	0	
aluresult	0000000	00000000
q1	0000000	0000FFFF, 000000FF
dataout	FFFFFFF	FFFFFFFF
mask	00	01, 10, 00
writedata	ABCD00	000000FF, ABCD0
flow	0040002	00400024, 00400028

Load from register 0.
Store in regiser 8

opcode for lb

Flow control. Note:
correct incrementation

Addr = r0 + immidiate offset

ALU set to add

immidiate value fed into ALU port B

Read from mem[0(r0)]

DM output masked
to only allow 8 LSB

Msgs

Lh Instructions

Signal	Msgs	Waveform values
rs	0	0
rt	8	8
rd	0	0
op	24	23, 25, 24
clk	1	
mclk	0	
rst	1	
instruction	900800(	8…, 94080000, 900800(
count	0040001	0040001C, 00400020
count1	0040001	00400020, 00400024
busa	0000000(	00000000
immediate	0000000(	0…, 00000000
aluop	010	010
alusrc	1	
aluresult	0000000(	0…, 00000000
q1	0000FFF	000000FF, 0000FFFF
dataout	FFFFFFF	000000FF, FFFFFFFF
mask	10	00, 01, 10
writedata	0000000	000000FF, 0000FFFF, 000000
flow	0040001	00400020, 00400024

Load from register 0.
Store in regiser 8

opcode for lh

Flow control. Note:
correct incrementation

Addr = r0 + immidiate offset

ALU set to add

immidiate value fed into ALU port B

Read from mem[0(r0)]

DM output masked
to only allow 16 LSB

Msgs

Lui Instruction

Signal	Msgs	Waveform values
rs	0	0
rt	0	8, 0
rd	0	21, 0
op	00	0F, 00
clk	1	
mclk	0	
rst	1	
instruction	0000000	3C08ABCD, 000000
count	004000	00400024, 00400028
count1	004000	00400028, 0040002C
busa	0000000	00000000
immediate	0000000	FFFFABCD, 000000
aluop	101	101
alusrc	0	
aluresult	0000000	00000000, 000000
q1	0000000	000000FF, 000000
dataout	FFFFFFF	FFFFFFFF
mask	00	00
writedata	0000000	ABCD0000, 000000
flow	004000	00400028, 0040002C

Load from register 0. Store in regiser 8

opcode for lui

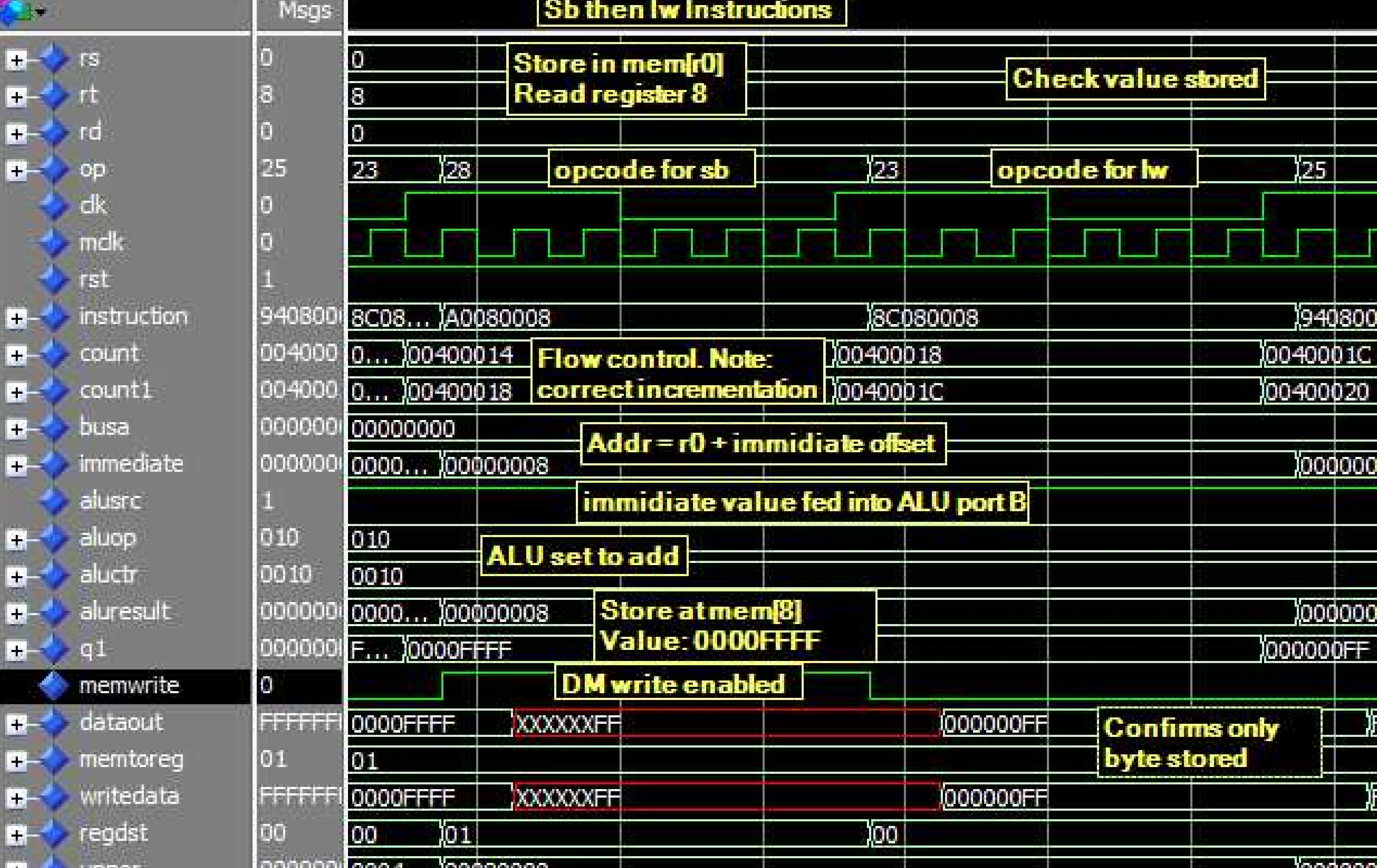
Answer stored on clk

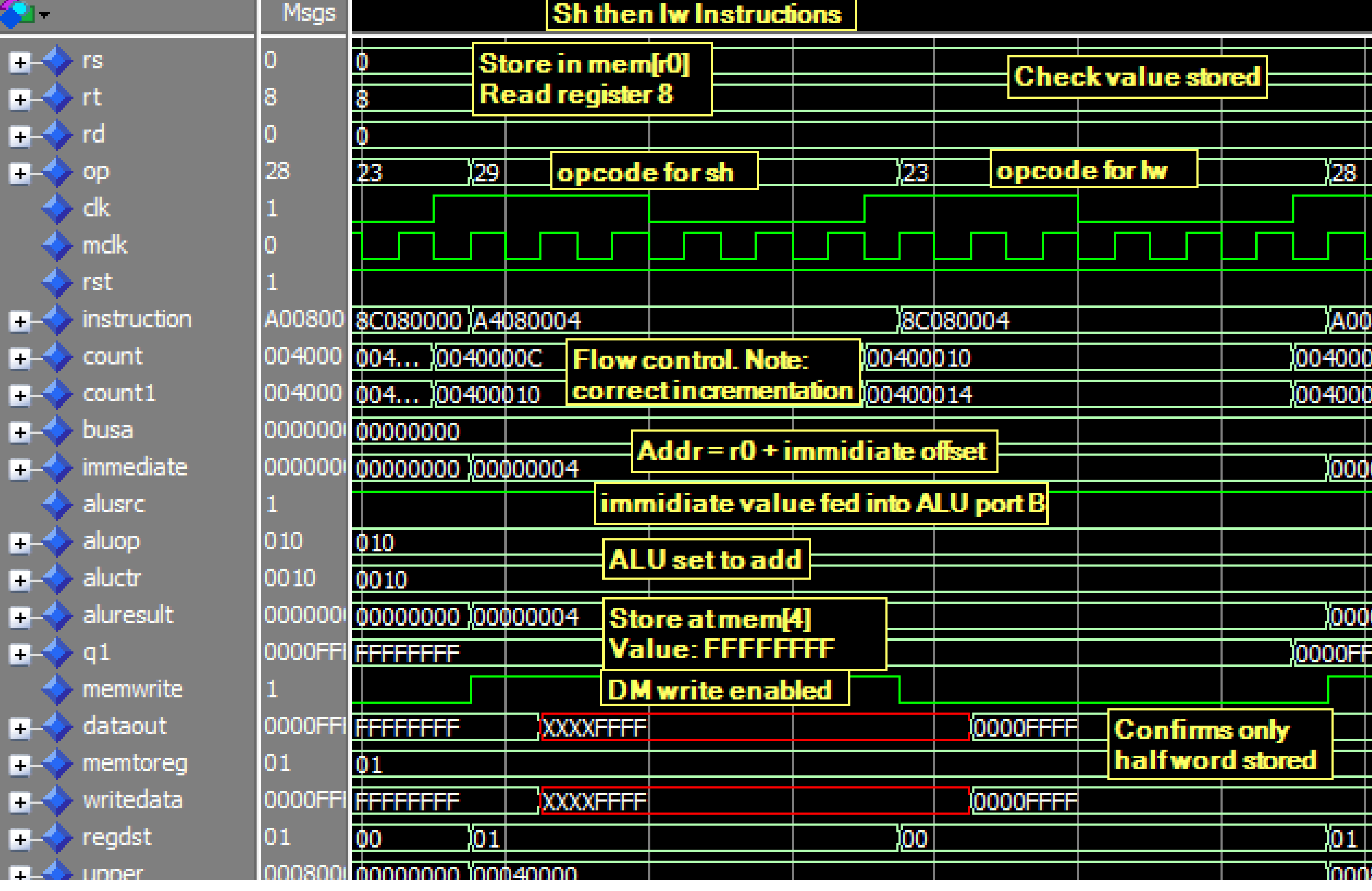
Flow control. Note: correct incrementation

Immidiate value to be used

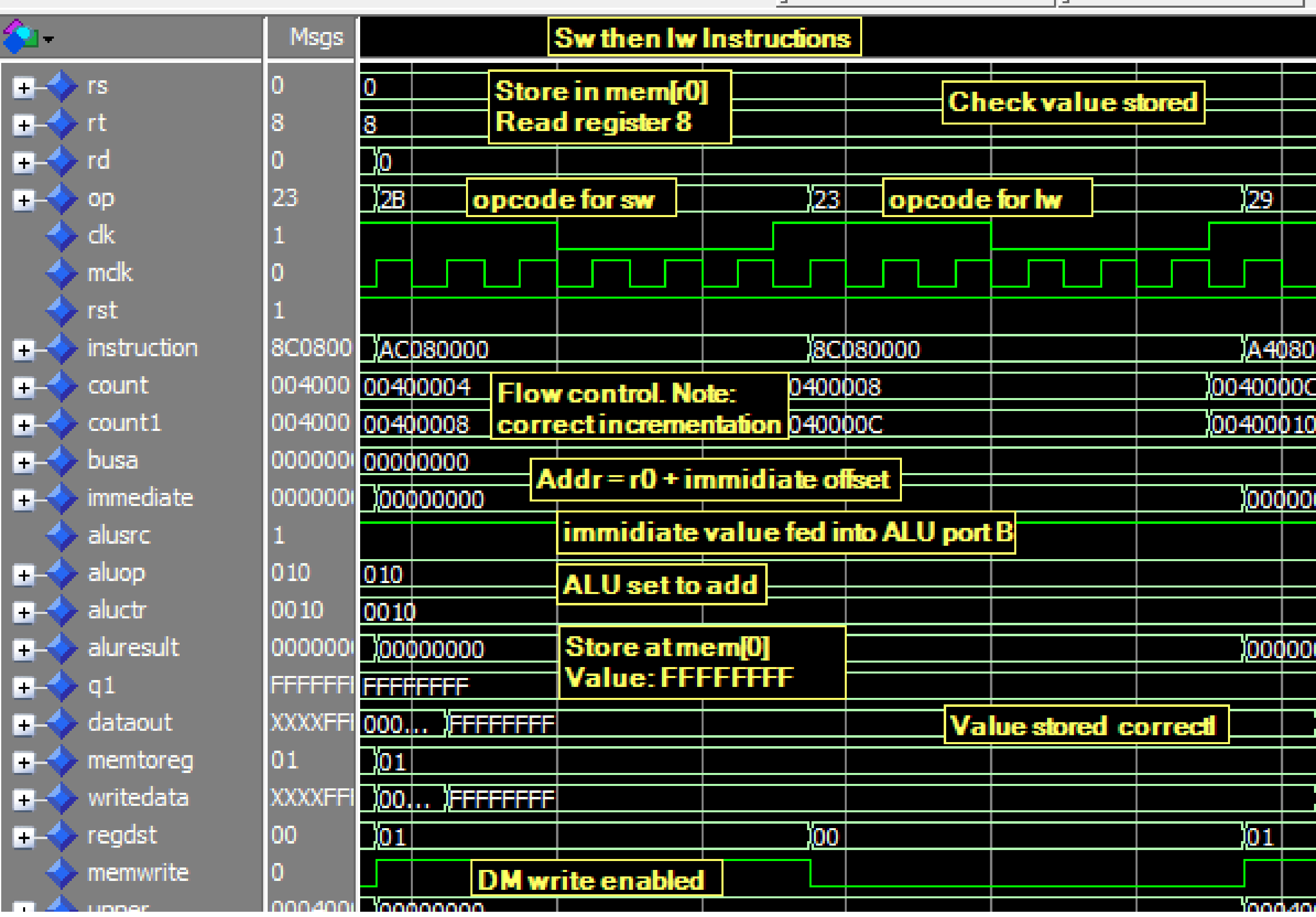
MemtoReg set to load modified immidiate value

Result: 16 LSB shifted to MSB


Msgs
Sb then lw Instructions
rs
rt
rd
op
clk
mclk
rst
instruction
count
count1
busa
immediate
alusrc
aluop
aluctr
aluresult
q1
memwrite
dataout
memtoreg
writedata
regdst
Store in mem[r0]
Read register 8
Check value stored
opcode for sb
opcode for lw
Flow control. Note: correct incrementation
Addr = r0 + immidiate offset
immidiate value fed into ALU port B
ALU set to add
Store at mem[8]
Value: 0000FFFF
DM write enabled
Confirms only byte stored



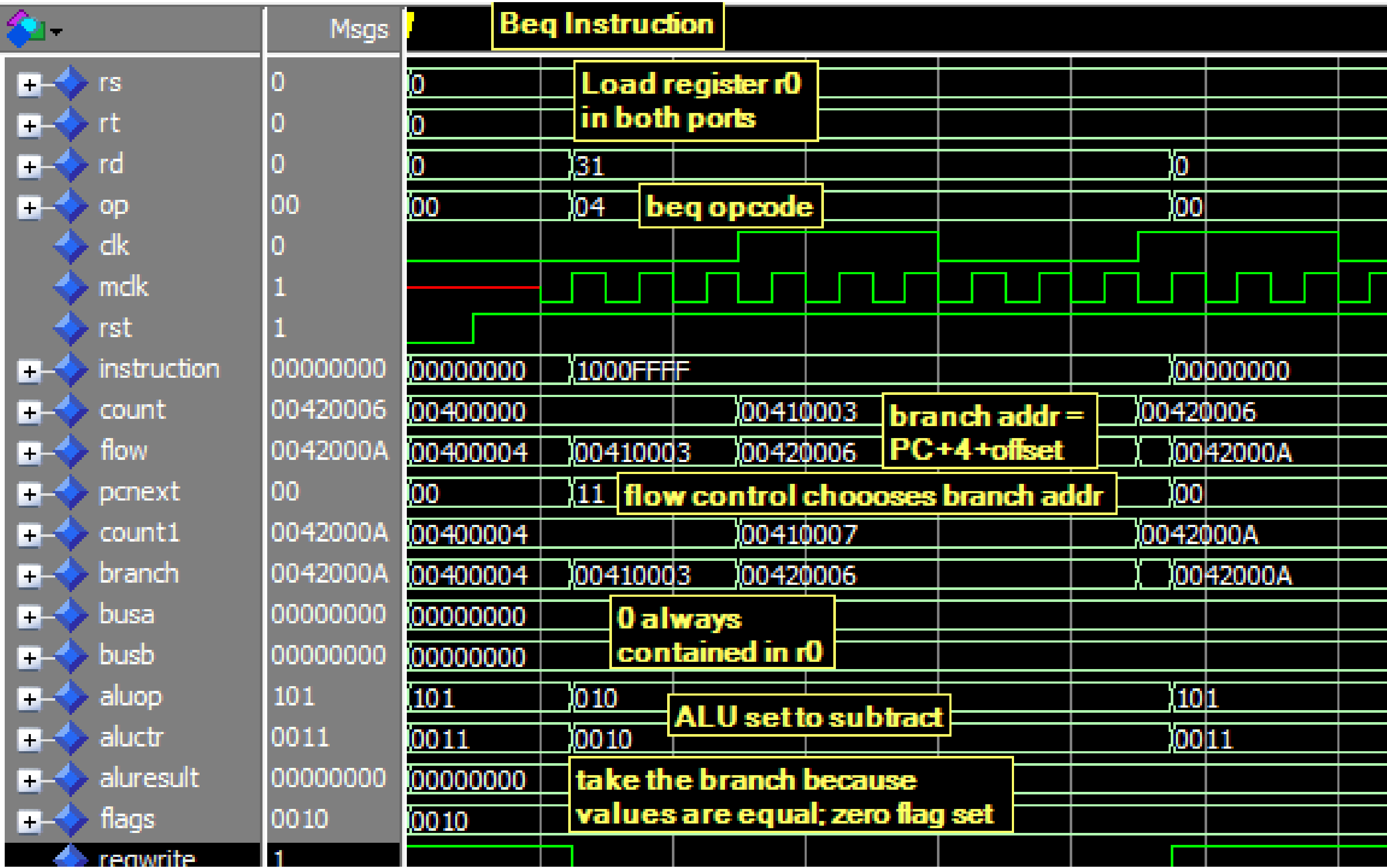
Msgs
Sh then lw Instructions
Store in mem[r0]
Read register 8
Check value stored
opcode for sh
opcode for lw
Flow control. Note:
correct incrementation
Addr = r0 + immidiate offset
immidiate value fed into ALU port B
ALU set to add
Store at mem[4]
Value: FFFFFFFF
DM write enabled
Confirms only
half word stored
rs
rt
rd
op
clk
mclk
rst
instruction
count
count1
busa
immediate
alusrc
aluop
aluctr
aluresult
q1
memwrite
dataout
memtoreg
writedata
regdst
upper



Msgs
Sw then lw Instructions
rs
rt
rd
op
clk
mclk
rst
instruction
count
count1
busa
immediate
alusrc
aluop
aluctr
aluresult
q1
dataout
memtoreg
writedata
regdst
memwrite
Store in mem[r0]
Read register 8
Check value stored
opcode for sw
opcode for lw
Flow control. Note:
correct incrementation
Addr = r0 + immidiate offset
immidiate value fed into ALU port B
ALU set to add
Store at mem[0]
Value: FFFFFFFF
Value stored correctl
DM write enabled

Branch Instructions:
Covers: beq, bne (I-type instructions completed)
Hardware: add32 for branch destination, larger program flow mux
Description: The branch instructions need special hardware affecting the flow control of the processor. First we add a 32-bit adder to sum the next address plus the 16-bit offset provided by the 16 LSB of the instruction. This branch destination is then input as a choice into the flow control mux. This mux must be expanded to a 4 to 1 mux and the control signal, Pcnext, must be made a 2-bit signal for this to work. Note that the mux currently has one unused input that we will use later. There are no other controls that need to be added because the zero flag is used as the other control signal. The other control signals affected are ALUop set to add, registerFile and DM write disabled, regDest set to load rs and rt, and ALUsrc set to load port B.

The following screen shots were captured from lab4_test.mif and lab4demo.mif for these two branch instructions.



Msgs
Beq Instruction
rs
rt
rd
op
clk
mclk
rst
instruction
count
flow
pcnext
count1
branch
busa
busb
aluop
aluctr
aluresult
flags
regwrite
Load register r0
in both ports
beq opcode
branch addr =
PC+4+offset
flow control chooses branch addr
0 always
contained in r0
ALU set to subtract
take the branch because
values are equal; zero flag set

Bne Instruction

Signal	Msgs	Waveform values
rs	1	1
rt	1	1, 0, 1
rd	31	31
op	08	08, 05, 08
clk	0	
mclk	0	
rst	1	
instruction	2021FFF	202..., 1420FFF9, 2021FFFF
count	0041003	00400038, 00410035
flow	0041003	00410035, 00410039
pcnext	00	00, 11, 00
count1	0041003	0040003C, 00410039
input2	0021FFF	002..., 0020FFF9, 0021FFFF
branch	0041003	00410035, 00410038
busa	0000000	00000003
busb	FFFFFFF	FFF..., 00000000, FFFFFFFF
immediate	FFFFFFF	FFF..., 0000FFF9, FFFFFFFF
aluop	010	010
aluctr	0010	0010
alusrc	1	
aluresult	0000000	00000003, 00000002
flags	0001	0001, 0000, 0001
regwrite	1	
dataout	0000000	00000000

Load values from registers 1 and 0

opcode for bne

loads branch addr

Compares 3 and 0

set ALU to subtract for comparison

ALU source B set to rt

result is not zero so branch is taken

J-type Instructions
Covers: j, jal
Hardware:address incrementation mux
Description: The first jump instruction does not require the addition of any extra hardware because there is already an open input slot into the 4-1 control flow mux. By simply connecting the jump destination provided by the 26 LSB of the instruction we can perform jump instructions. However, to implement the jump and link instruction we must add hardware. Because the value stored as the return address is incremented by 8 instead of 4 we add a 2-1 mux going into the PC incrementation adder. The control signal step alows us to choose a step size as 4 or 8 now. Other than that these two instructions have the same control logic that is as follows: set PCnext to load the jump destination, set registerFile to write only for jal, DM write disabled, only for jal MemtoReg is set to the incremented address.

The following screen shots were captured from lab4_test.mif and lab4demo.mif.

Instruction	Opcode rs rt rd shamt funct	
jal	0000.11 00.0000.0000.0000.0000.0010.0100	
jr	0000.00 11.111 0.0000. 0000.0 000.00 00.1000	(for ra verification)

Jal Instruction

Signal	Msgs	Waveform values
rs	11111	00000, 11111
rt	00000	00000
rd	00000	00000
op	000000	000000, 000011, 000000
clk	0	
mclk	1	
rst	1	
instruction	03E00008	00000000, 0C00003F, 03E00008
count	0000003F	00400000, 0000003F
pcnext	01	00, 10, 01
busa	00400008	00000000, 00400008
flow	00400008	00400004, 0000003F, 00400008
count1	00000043	00400004, 00400008, 00..., 00000043
immediate	00000008	00000000, 0000003F, 00000008
branch	0000004B	00400004, 00400047, 00..., 0000004B
writedata	00000000	00000000, 00400008, 00..., 00000000
destination	00000	00000, 11111, 00000
regwrite	1	
upper	00080000	00000000, 003F0000, 00080000

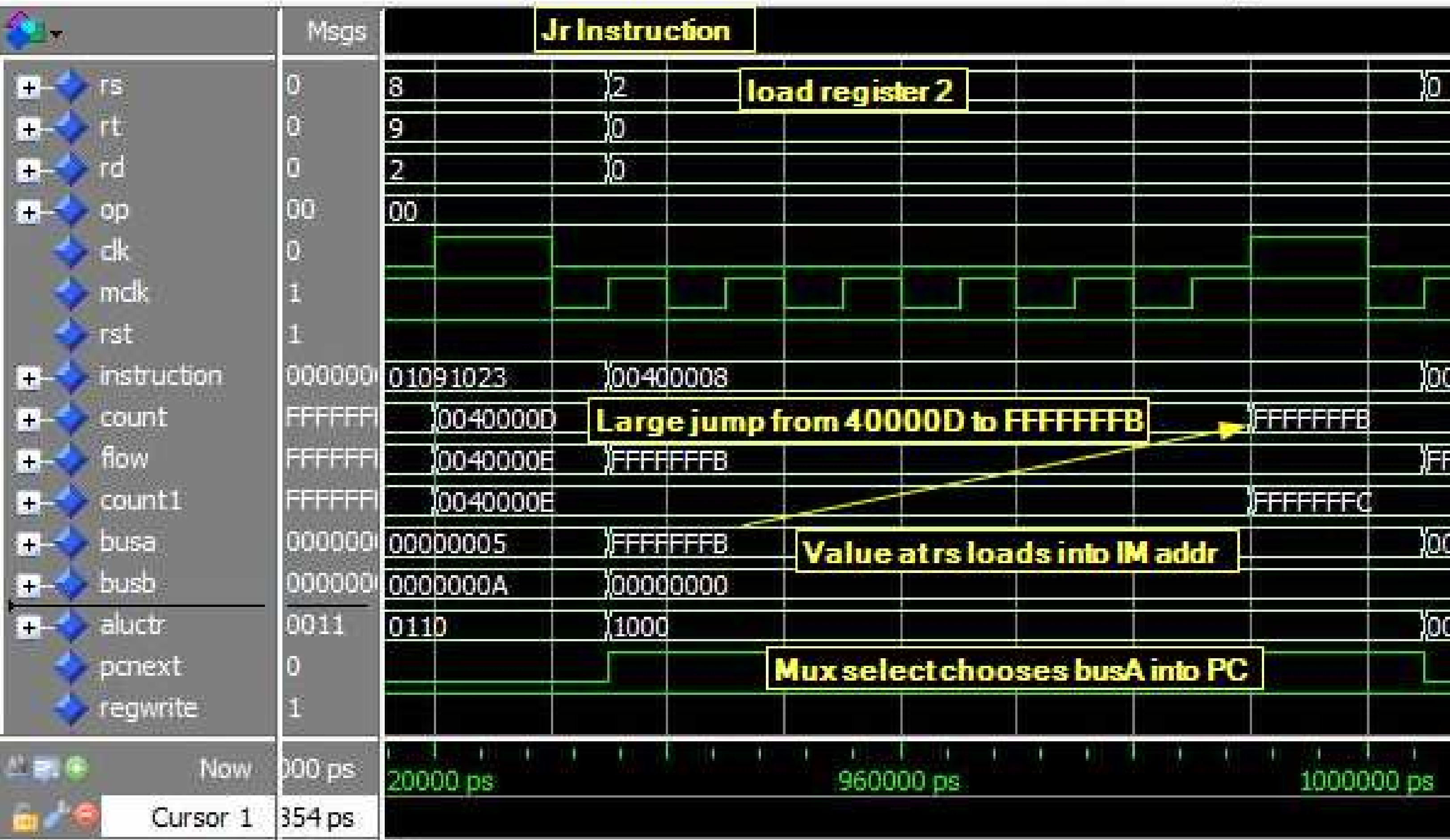
jal opcode

jr instruction confirms ra correct

jump address

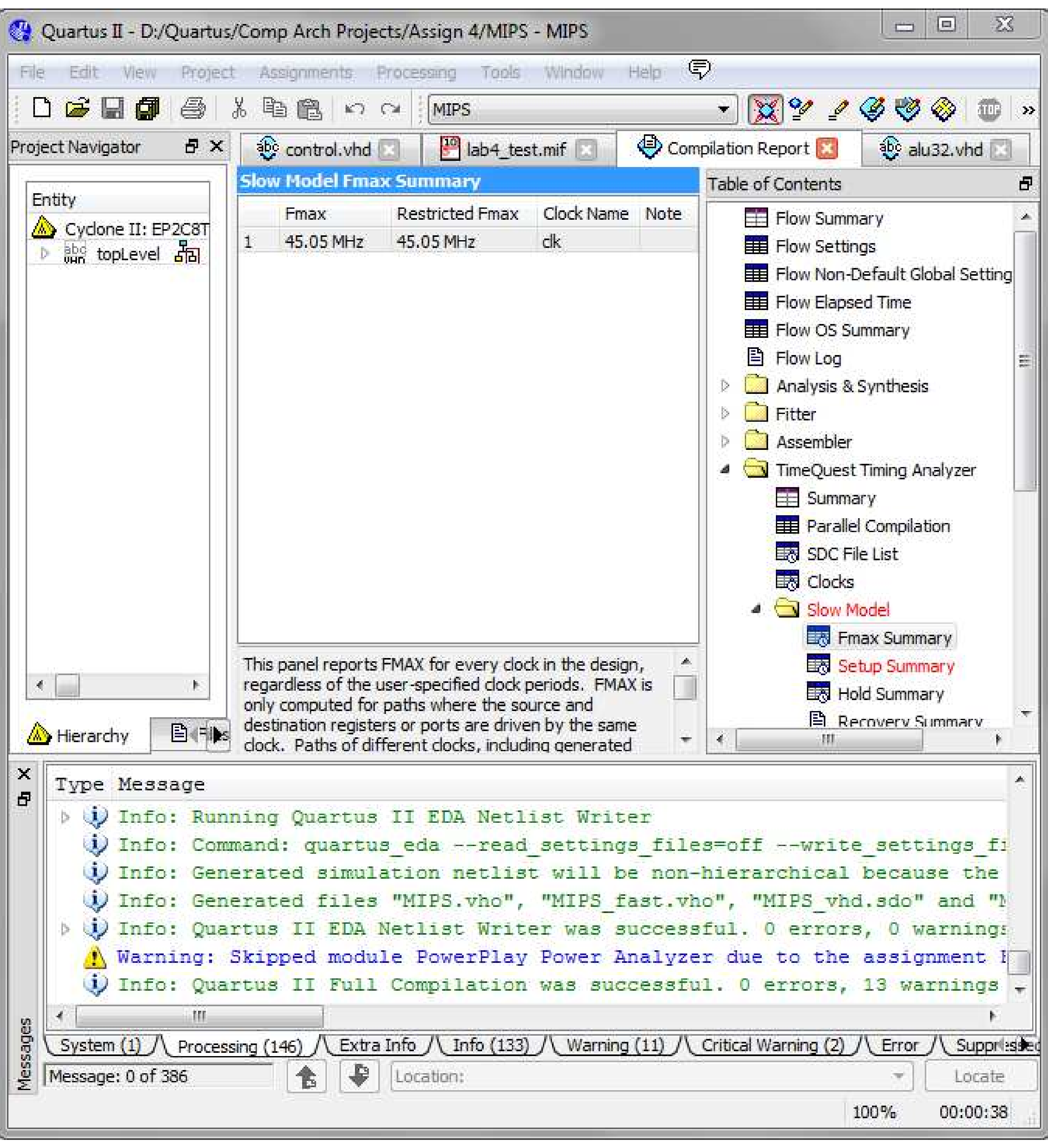
set to load instruction[20-0]

Write PC+8 to R[31]


Jr Instruction
load register 2
Large jump from 40000D to FFFFFFFB
Value at rs loads into IM addr
Mux select chooses busA into PC

Timing Analysis:
A screen shot of the final timing analysis may be seen on the next page. The final Fmax u 45.05 MHz. This clk rate is dictated by the critical path which is the data memory load instructions. The path starts from the PC and IM and proceeds through the register file and control logic, through the ALU to calculate the DM address, retrieves the values from DM, then through tw0 4-1 muxes and finally into the registerFile to be stored. This critical path is almost entirely optimized. The only room for improvement that I could see was to use tri-states instead of 4-1 muxes. However, after doing a timing simulation, I found that the tri-states were only negligibly faster (order of pico seconds) and not worth changing out.

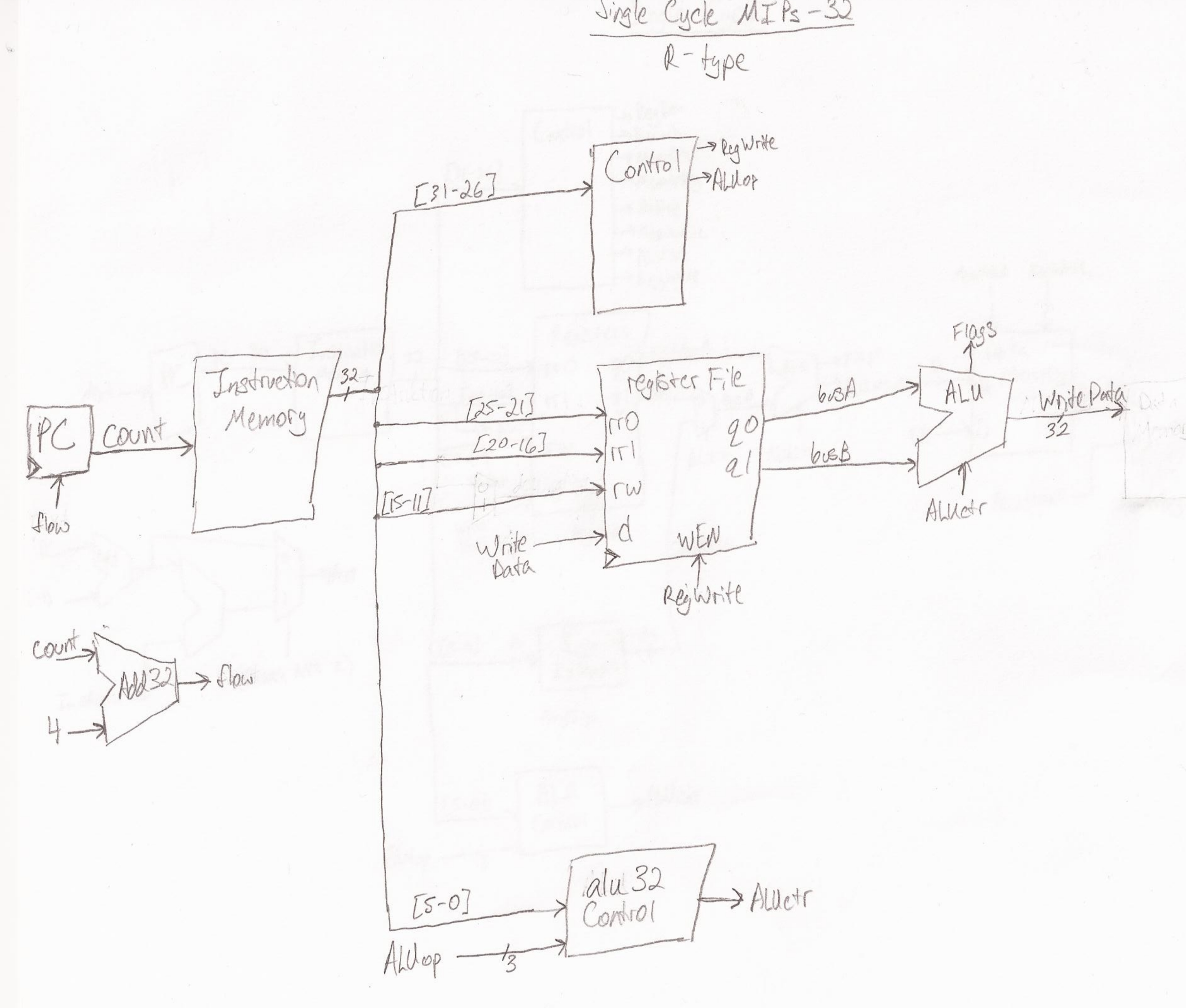
The pages after the timing analysis show the very most basic block diagram evolving to the complete block diagram for the MIPS processor followed by my code for the top_level and control logic.



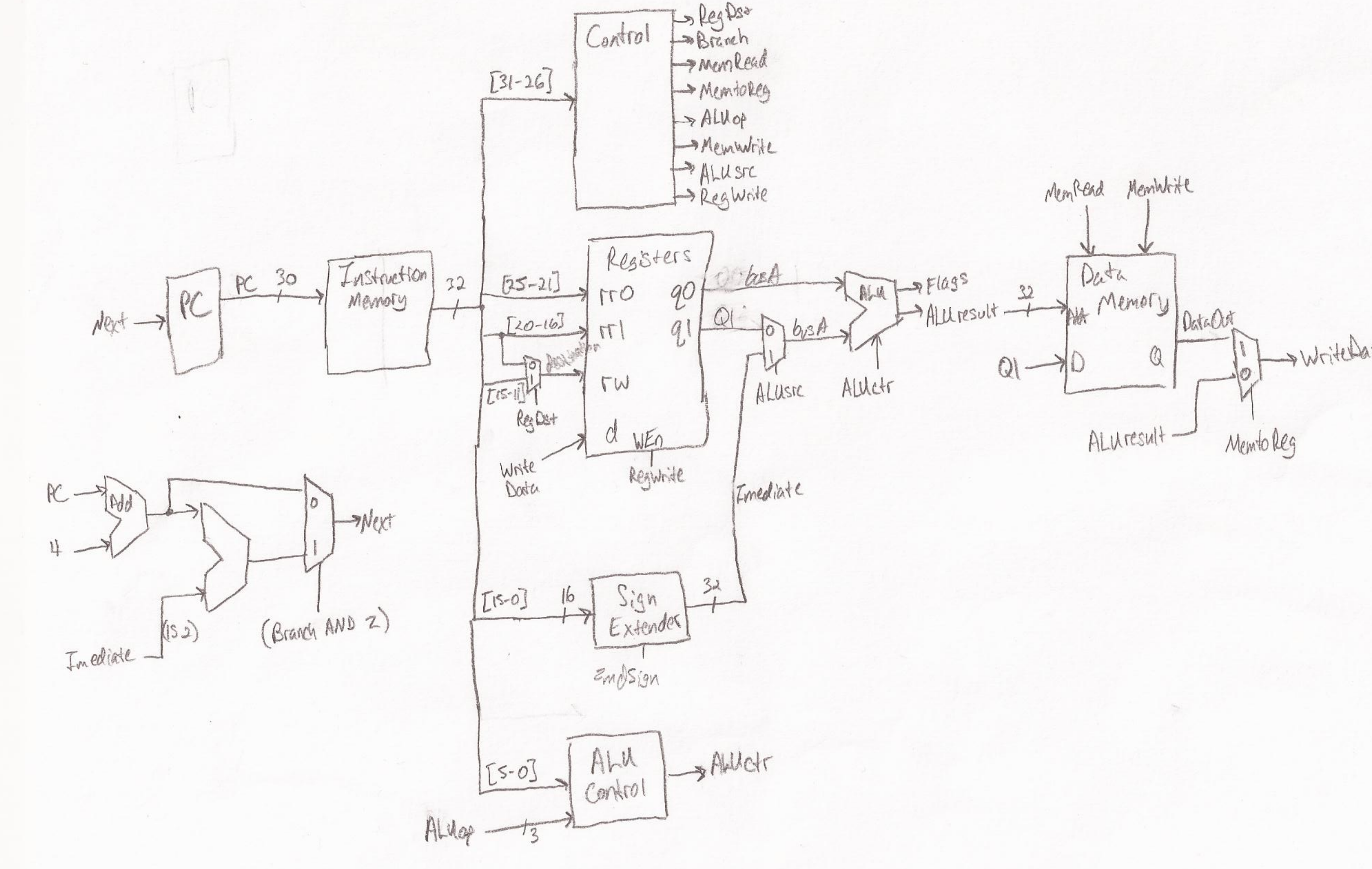
Quartus II - D:/Quartus/Comp Arch Projects/Assign 4/MIPS - MIPS
File Edit View Project Assignments Processing Tools Window Help
MIPS
Project Navigator
control.vhd
lab4_test.mif
Compilation Report
alu32.vhd
Entity
Cyclone II: EP2C8T
topLevel
Hierarchy
Slow Model Fmax Summary
Fmax Restricted Fmax Clock Name Note
1 45.05 MHz 45.05 MHz clk
This panel reports FMAX for every clock in the design, regardless of the user-specified clock periods. FMAX is only computed for paths where the source and destination registers or ports are driven by the same clock. Paths of different clocks, including generated
Table of Contents
Flow Summary
Flow Settings
Flow Non-Default Global Setting
Flow Elapsed Time
Flow OS Summary
Flow Log
Analysis & Synthesis
Fitter
Assembler
TimeQuest Timing Analyzer
Summary
Parallel Compilation
SDC File List
Clocks
Slow Model
Fmax Summary
Setup Summary
Hold Summary
Recovery Summary
Type Message
Info: Running Quartus II EDA Netlist Writer
Info: Command: quartus_eda --read_settings_files=off --write_settings_fi
Info: Generated simulation netlist will be non-hierarchical because the
Info: Generated files "MIPS.vho", "MIPS_fast.vho", "MIPS_vhd.sdo" and "M
Info: Quartus II EDA Netlist Writer was successful. 0 errors, 0 warnings
Warning: Skipped module PowerPlay Power Analyzer due to the assignment E
Info: Quartus II Full Compilation was successful. 0 errors, 13 warnings
System (1)
Processing (146)
Extra Info
Info (133)
Warning (11)
Critical Warning (2)
Error
Messages
Message: 0 of 386
Location:
Locate
100%
00:00:38

Single Cycle MIPs-32

R-type



Single Cycle



Single Cycle MIPs-32

Complete

PC
count
flow
Instruction Memory
32
[31-26]
[25-21]
[20-16]
[15-11]
[15-0]
[5-0]
Control
RegWrite
ALUop
ImdSign
ALUsrc
RegDest
PCnext
byteEna
register File
rr0
rr1
rw
d
WEN
q0
q1
31
destination
RegDest
Write Data
RegWrite
busA
busB
Q1
ALUsrc
ALU
Flags
ALUctr
ALUresult
32
MemRead
MemWrite
Data Memory
D
DataOut
Q1
byteEna
Sign Extender
16
32
ImdSign
Immediate
alu32 Control
ALUop
3
ALUctr
count
Add32
count1
4
8
step
busA
instruction[25-0]
branch
00
01
10
11
flow
PCnext
count1
instruction[15-0]
Add32
branch
DataOut
0x0000 & DataOut
0x000000 & DataOut
mask
maskData
ALUresult
count1
instruction[15-0]&0
MemtoRes
WriteData

```
1   -- Zack Smaridge
2   -- Computer Architecture, UF
3   -- Spring 2012
4   -- Top Level Organization
5
6   library ieee;
7   use ieee.std_logic_1164.all;
8   use ieee.numeric_std.all;
9
10  entity topLevel is
11
12     port(
13        clk, mclk, rst     : in std_logic;
14        rs, rt, rd         : out std_logic_vector(4 downto 0);
15        op                 : out std_logic_vector(5 downto 0));
16
17  end topLevel;
18
19
20  architecture Behavior of topLevel is
21
22  signal count, count1, busA, busB, Q1       : std_logic_vector(31 downto 0);     -- data busses
23  signal immediate, ALUresult, instruction   : std_logic_vector(31 downto 0);     -- data buses
24  signal DataOut, flow, branch, input2       : std_logic_vector(31 downto 0);     -- data buses
25  signal WriteData, upper, increment         : std_logic_vector(31 downto 0);     -- data buses
26  signal halfMask, byteMask, maskData        : std_logic_vector(31 downto 0);     -- data buses
27  signal ALUctr, flags, byteEna              : std_logic_vector(3 downto 0);      --
    intermediate controls
28  signal destination, destination1           : std_logic_vector(4 downto 0);      --
    intermediate data bus
29  signal RegWrite, ImdSign, Shdir            : std_logic;       -- direct control signals
30  signal ALUsrc, MemWrite, step              : std_logic;       -- direct control signals
31  signal ALUop                               : std_logic_vector(2 downto 0);      -- direct
    control signals
32  signal PCnext, RegDst, MemtoReg, mask      : std_logic_vector(1 downto 0);      -- control
    signal based on ALUctr
33
34
35  begin
36
37  Shdir    <= instruction(1);    -- shift bit ctly corresponds to func field
38
39  -- CONTROLLER
40  Brain: entity work.control
41     port map(
42        instruction     => instruction(31 downto 26),
43        funct           => instruction(5 downto 0),
44        Zflag           => flags(1),
45        RegWrite        => RegWrite,
46        ImdSign         => ImdSign,
47        ALUop           => ALUop,
48        ALUsrc          => ALUsrc,
49        step            => step,
50        MemWrite        => MemWrite,
51        byteEna         => byteEna,
52        mask            => mask,
53        PCnext          => PCnext,
54        RegDst          => RegDst,
55        MemtoReg        => MemtoReg);
56
57  -- Program Counter
58  u_PC: entity work.reg32
```

```
59        port map(
60           D      => flow,
61           CLK    => clk,
62           Wr     => '1',
63           Clr    => rst,
64           Q      => count);
65
66     -- MUX for incrementing count by 4 or 8
67     u_stepper: entity work.mux32
68        port map(
69           input0    => x"00000004",
70           input1    => x"00000008",
71           sel       => step,
72           output    => increment);
73
74     -- Adder used exclusively for Program Counter
75     u_adder: entity work.add32
76        port map(
77           in0    => count,
78           in1    => increment,
79           sum    => count1);
80
81     --count11    <=    count1(31 downto 2) & "00";
82
83     -- Adder used exclusively for Program Counter
84     u_adder1: entity work.add32
85        port map(
86           in0    => count1,
87           in1    => immediate,
88           sum    => branch);
89
90     -- 32-bit MUX for flow control
91     u_floctr: entity work.mux432
92        port map(
93           input0    => count1,
94           input1    => busA,
95           input2    => input2,
96           input3    => branch,
97           sel       => PCnext,
98           output    => flow);
99
100    -- Instruction memory for program code
101    u_InstructionMemory: entity work.InstructionMemory
102       port map(
103          address   => count(9 downto 2),
104          clock     => mclk,
105          data      => (others => '0'),
106          wren      => '0',
107          q         => instruction);
108
109    -- Mux going into register file, chooses between rt and rd
110    u_registerMux: entity work.mux5
111       port map(
112          input0    => instruction(20 downto 16),
113          input1    => instruction(15 downto 11),
114          sel       => RegDst(0),
115          output    => destination1);
116
117    -- Mux going into register file, chooses between 1st mux and $ra
118    u_registerMux1: entity work.mux5
119       port map(
120          input0    => destination1,
```

```
121         input1    => "11111",
122         sel       => RegDst(1),
123         output    => destination);
124
125   -- Register file
126   u_registerFile: entity work.registerFile
127      port map(
128         rr0    => instruction(25 downto 21),
129         rr1    => instruction(20 downto 16),
130         rw     => destination,
131         d      => WriteData,
132         clk    => clk,
133         wr     => RegWrite,
134         rst    => rst,
135         q0     => busA,
136         q1     => Q1);
137
138   -- Extender for 16-bit immediate values
139   -- control signal ImdSign chooses zero or sign extension
140   u_extender: entity work.extender
141      port map(
142         in0    => instruction(15 downto 0),
143         Sel    => ImdSign,
144         out0   => immediate);
145
146   -- Decoder for two different signals to control ALU
147   u_ALUcontrol: entity work.alu32control
148      port map(
149         func   => instruction(5 downto 0),
150         ALUop  => ALUop,
151         control   => ALUctr);
152
153   -- 32-bit MIPS ALU
154   u_ALU: entity work.alu32
155      port map(
156         ia     => busA,
157         ib     => busB,
158         control   => ALUctr,
159         shamt     => instruction(10 downto 6),
160         shdir     => Shdir,
161         o      => ALUresult,
162         C      => flags(0),
163         Z      => flags(1),
164         S      => flags(2),
165         V      => flags(3));
166
167   -- Mux going into ALU ib
168   -- chooses beteen register and immediate value
169   u_ALUregister: entity work.mux32
170      port map(
171         input0    => Q1,
172         input1    => immediate,
173         sel       => ALUsrc,
174         output    => busB);
175
176   -- Data Memory, starting at
177   u_DataMemory: entity work.DataMemory
178      port map(
179         address   => ALUresult(7 downto 0),
180         byteena   => byteEna,
181         clock     => mclk,
182         data      => Q1,
```

```
183         wren      => MemWrite,
184         q         => DataOut);
185
186   halfMask <= x"0000" & DataOut(15 downto 0);
187   byteMask <= x"000000" & DataOut(7 downto 0);
188
189   -- 32-bit MUX for dataOut mask
190   u_datactr: entity work.mux432
191      port map(
192         input0    => DataOut,
193         input1    => halfMask,
194         input2    => byteMask,
195         input3    => DataOut,
196         sel       => mask,
197         output    => maskData);
198
199   -- 32-bit MUX for data control
200   u_datactr1: entity work.mux432
201      port map(
202         input0    => ALUresult,
203         input1    => maskData,
204         input2    => count1,
205         input3    => upper,
206         sel       => MemtoReg,
207         output    => WriteData);
208
209   op <= instruction(31 downto 26);
210   rs <= instruction(25 downto 21);
211   rt <= instruction(20 downto 16);
212   rd <= instruction(15 downto 11);
213
214   input2    <= "000000" & instruction(25 downto 0);
215   upper     <= instruction(15 downto 0) & x"0000";
216
217   end behavior;
218
219
220
221
```

```
1   -- Zack Smaridge
2   -- Computer Architecture, UF
3   -- Spring 2012
4   -- Controller
5
6   library ieee;
7   use ieee.std_logic_1164.all;
8   use ieee.numeric_std.all;
9
10  entity control is
11
12     port( instruction          : in std_logic_vector(5 downto 0);  -- instruction[31-26]
13           funct                : in std_logic_vector(5 downto 0);  -- ALU opcode
14           Zflag                : in std_logic;                     -- zero flag from ALU
15           RegWrite             : out std_logic;                    -- for Register File
16           ImdSign              : out std_logic;                    -- for extender and shifter
17           ALUop                : out std_logic_vector(2 downto 0); -- for ALU control decoder
18           ALUsrc, step         : out std_logic;                    -- salects inputb to ALU
19           MemWrite             : out std_logic;                    -- for data memory
20           byteEna              : out std_logic_vector(3 downto 0); -- for data memory
21           mask                 : out std_logic_vector(1 downto 0); -- for dataOut mask
22           PCnext, RegDst, MemtoReg    : out std_logic_vector(1 downto 0));-- controls program
    flow
23
24  end control;
25
26
27  architecture Behavior of control is
28
29  begin
30
31  controls: process(instruction, Zflag, funct)
32  begin
33     -- Set default values
34     RegDst   <= "01";    -- rw gets rd
35     RegWrite <= '1';     -- register file loads
36     ImdSign  <= '1';     -- sign extend immidiate values
37     ALUsrc   <= '0';     -- busB into ALU gets Q1 from registers
38     MemWrite <= '0';     -- data write enable off
39     MemtoReg <= "00";    -- ALUresult goes to registerFile
40     ALUop    <= "101";   -- defaults to R-types
41     byteEna  <= "1111";  -- all 4-bytes enabled
42     PCnext   <= "00";    -- PC++
43     step     <= '0';     -- increments PC by 4
44     mask     <= "00";    -- DataOut unmasked
45
46     case instruction is
47        when "000000"  =>    -- R-type instructions
48           -- optimize the common case
49           if(funct = "001000") then
50              PCnext <= "01";
51           end if;
52
53        when "001000"  =>    -- addi
54           ALUsrc   <= '1';     -- select immediate value
55           --ImdSign   <= '1';      -- select signed numbers
56           ALUop    <= "010";   -- select ALU addition
57           RegDst(0)<= '0';     -- select rt as destination
58
59        when "001001"  =>    -- addiu
60           ALUsrc   <= '1';     -- select immediate value
61           ImdSign  <= '0';     -- select unsigned numbers
```

```
 62             ALUop     <= "010";    -- select ALU addition
 63             RegDst(0)<= '0';       -- select rt as destination
 64
 65          when "001100"   =>     -- andi
 66             ALUsrc    <= '1';      -- select immediate value
 67             ImdSign   <= '0';      -- select unsigned numbers
 68             ALUop     <= "000";    -- select ALU AND
 69             RegDst(0)<= '0';       -- select rt as destination
 70
 71          when "000100"   =>     -- beq
 72             RegWrite  <= '0';      -- disable writing to register file
 73             ImdSign   <= '0';      -- select unsigned numbers
 74             ALUop     <= "010";    -- ALU addition
 75             if(Zflag = '1') then
 76                PCnext  <= "11";    -- branch address load into PC
 77             end if;
 78
 79          when "000101"   =>     -- bne
 80             RegWrite  <= '0';      -- disable writing to register file
 81             ImdSign   <= '0';      -- select unsigned numbers
 82             ALUop     <= "010";    -- ALU addition
 83             if(Zflag = '0') then
 84                PCnext  <= "11";    -- branch address load into PC
 85             end if;
 86
 87          when "000010"   =>     -- j
 88             RegWrite  <= '0';      -- disable writing to register file
 89             PCnext    <= "10";     -- jump address loaded into PC
 90
 91          when "000011"   =>     -- jal
 92             step      <= '1';      -- implements PC by 2 instead of 1
 93             RegDst(1)<= '1';       -- register file loads into $ra
 94             PCnext    <= "10";     -- jump address loaded into PC
 95             MemtoReg  <= "10";     -- registerFile chooses PC+8
 96
 97          when "100100"   =>     -- lbu
 98             ALUop     <= "010";    -- select ALU addition
 99             ALUsrc    <= '1';      -- select immediate value
100             byteEna   <= "0001";   -- only store 2 bytes
101             MemtoReg  <= "01";     -- select DataOut into registerFile
102             RegDst(0)<= '0';       -- select rt as destination
103             mask      <= "10";     -- byte mask
104
105          when "100101"   =>     -- lhu
106             ALUop     <= "010";    -- select ALU addition
107             ALUsrc    <= '1';      -- select immediate value
108             byteEna   <= "0011";   -- only store 2 bytes
109             MemtoReg  <= "01";     -- select DataOut into registerFile
110             RegDst(0)<= '0';       -- select rt as destination
111             mask      <= "01";     -- half word mask
112
113          when "001111"   =>     -- lui
114             --ALUsrc <= '1';       -- select immediate value
115             -- ***** More Hardware Needed *****
116             RegDst    <= "00";     -- rw gets rt
117             MemtoReg  <= "11";     -- register file loads immediate
118
119          when "100011"   =>     -- lw
120             ALUop     <= "010";    -- select ALU addition
121             ALUsrc    <= '1';      -- select immediate value
122             MemtoReg  <= "01";     -- select DataOut into registerFile
123             RegDst(0)<= '0';       -- select rt as destination
```

```
124
125         when "001101"  =>    -- ori
126            ALUsrc    <= '1';     -- select immediate value
127            ImdSign   <= '0';     -- select unsigned numbers
128            ALUop     <= "110";   -- select ALU OR
129            RegDst(0)<= '0';      -- select rt as destination
130
131         when "001010"  =>    -- slti
132            ALUsrc    <= '1';     -- select immediate value
133            --ImdSign   <= '1';      -- select signed numbers
134            ALUop     <= "111";   -- select ALU set less than signed
135            RegDst(0)<= '0';      -- select rt as destination
136
137         when "001011"  =>    -- sltiu
138            ALUsrc    <= '1';     -- select immediate value
139            ImdSign   <= '0';     -- select unsigned numbers
140            ALUop     <= "100";   -- select ALU set less than unsigned
141            RegDst(0)<= '0';      -- select rt as destination
142
143         when "101000"  =>    -- sb
144            ALUop     <= "010";   -- select ALU addition
145            ALUsrc    <= '1';     -- select immediate value
146            byteEna   <= "0001";  -- only store 1 byte
147            MemtoReg  <= "01";    -- select DataOut into registerFile
148            MemWrite  <= '1';     -- RAM write enable
149            RegWrite  <= '0';     -- disable writing to register file
150
151         when "101001"  =>    -- sh
152            ALUop     <= "010";   -- select ALU addition
153            ALUsrc    <= '1';     -- select immediate value
154            byteEna   <= "0011";  -- only store 2 bytes
155            MemtoReg  <= "01";    -- select DataOut into registerFile
156            MemWrite  <= '1';     -- RAM write enable
157            RegWrite  <= '0';     -- disable writing to register file
158
159         when "101011"  =>    -- sw
160            ALUop     <= "010";   -- select ALU addition
161            ALUsrc    <= '1';     -- select immediate value
162            MemtoReg  <= "01";    -- select DataOut into registerFile
163            MemWrite  <= '1';     -- RAM write enable
164            RegWrite  <= '0';     -- disable writing to register file
165
166         when others    =>
167
168         end case;
169
170   end process;
171
172   end behavior;
173
174
175
176
```