CS152: Computer Architecture and Engineering

Fall, 2000

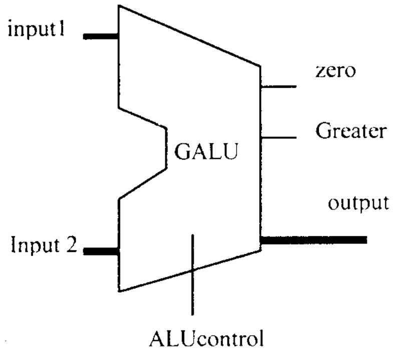
Midterm 1

Professor W. Brodersen

Problem 1: Single-cycle Processor Performance (35 points)

We have a single cycle processor as we learned in class. Besides the instructions R-format, lw, sw, and beq that the processor already implemented, the designer figured that certain software does the following instructions extensively

cs $t $sl $s2 **cslw** $t $sl $s2 **cslwcs** $t $sl $s2

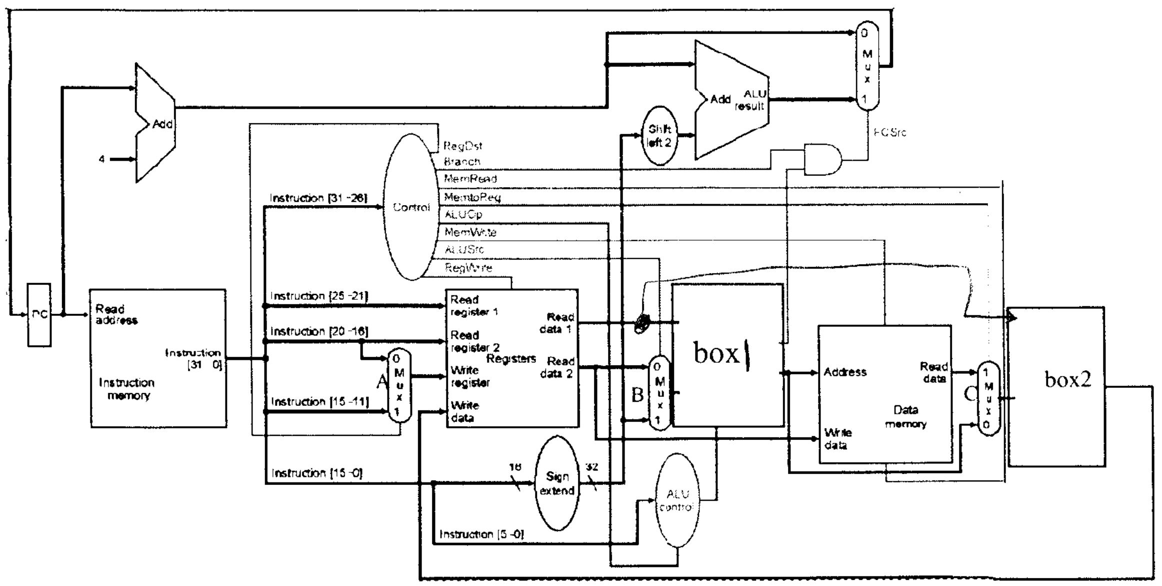


Besides the usual PC handling and instruction fetch as before,

- the **cs** instruction is described as:
  $t <= max($sl, $s2)
- The **cslw** does the following:
  $t <=memory [max($sl,$s2)]
- The **cslwcs** instruction the following:
  $t <= max{$sl, memory[max ($s1, $s2)]}

So now you see that **cs** and **1w** in above names stand for "compare select" and "load word".

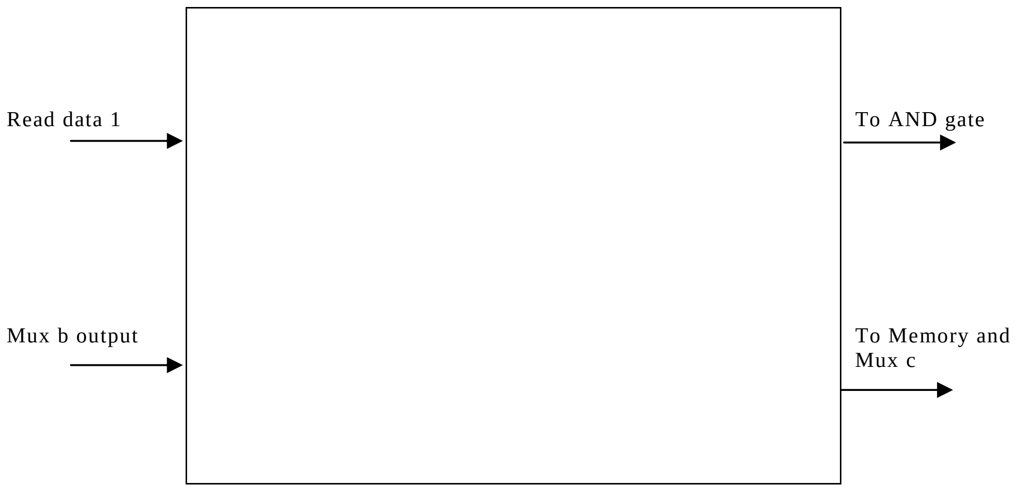
The designer plans to modify the datapath and control signal based on the processor showed in next page. It's a textbook processor with some missing blocks and wiring to fill in. The components that can be used are any of the blocks ***already present*** in the design and the generalized GALU (showed above) with an additional output GREATER that equals to 1 if Input1-Input2 >0, and equals to 0 if Input1-Input2 =< 0.



A) [10 points] ***Describe below in words and draw in box* 1** the blocks that need to be added in order to implement the **cs** and **cslw** instructions. ***Furthermore, draw in box 2*** any additional blocks to implement the **cslwcs** instruction. *(Hint: think about doing so by adding GALU(s), MUX(es), AND(s), and possibly new control signal(s))*

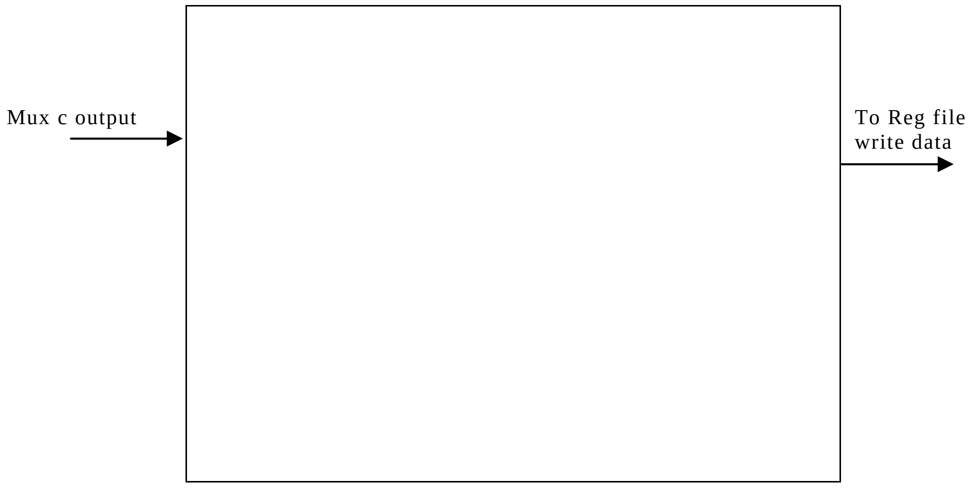
i) For **cs** and **cslw:**

Box 1



ii) For **cslwcs:**

Box 2



B) [7 points] Fill in the control table below with X, 0 or 1. Add in new columns) if you need new control signal(s). You may not need all three columns.

Instruction	ALUsrc	Mem-ToReg	RegWr	Mem-Read	Mem-Write			
SW	1		0	0	1			
R-format	0	0		0				
cslw								
cslwcs								

C) [9 points] You will point out the ***critical path,*** calculate the ***critical path delay in ns,*** and hence determine the ***fastest clock rate in MHz***. Finally, you will use some program features to ***calculate the execution tine of the processor.*** Use the following delay parameters (ignore hold time and regard the register delay as clock-to-Q):

- ALU or GALU, delay = 15 (ns)
- Sign/zero extender, delay = 3(ns)
- 2-1 Mux, delay = 2 (ns)
- Memory, delay (both write and read) = 10 (ns)
- Register, delay = 1 (ns)
- Register files, read delay =10 (ns), write setup time=10 (ns)

All other components' delays are ignored (i.e. zero).

i) The critical, path is (write down the components' names in order):

ii) The critical path delay in ns and the fastest clock rate in MHz:

iii) Now use a program to evaluate the performance of the processor. The program has 10% **lw** instructions, 10% **beq**, 10% **sw**, 20% R-format, 25% **cs**, 15% **cslw**, 10% **cslwcs**. The program contains 1,000,000 instructions in total. How much time is needed to run this program?

D) [9 points] ***Please write down as simply as possible the decomposition*** of the **cslwcs** instruction in terms of **cs, cslw, 1w, sw** and/or **R-format** instructions. Thus the original processor can be modified only to run additional **cs** and **cslw** instructions in one cycle and use multiple cycles for **cslwcs.** ***Estimate the critical path delay*** for this processor, and hence ***evaluate the time needed*** to run the same program in part C.iii above.

i) Define cslwcs $t $sl $s2, in terms of cs, cslw, lw, sw and R-format instructions:

ii) The new critical path delay:

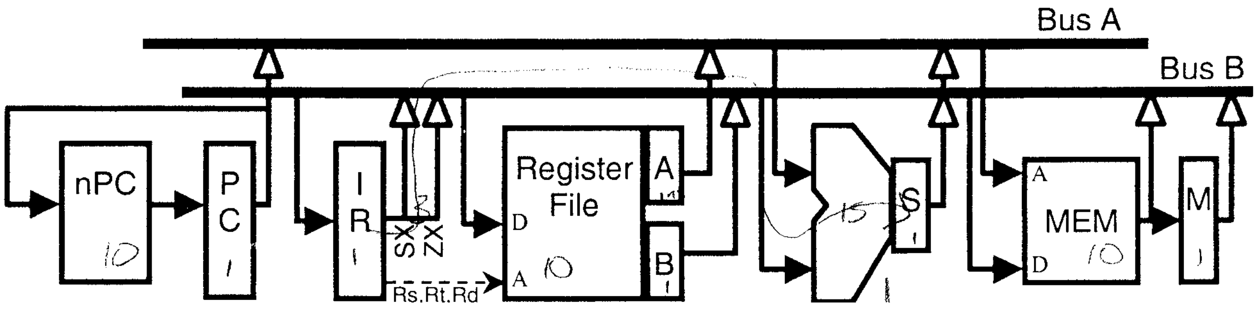
iii) The new execution time of the program:

Problem 2: Bus-based Multicycle Processor (30 points)

The datapath below forms a multicycle processor which uses two time-multiplexed buses for communication rather than point-to-point connections and muxes. What are the pros and cons of such an architecture? Is it a good idea? This problem will get you started toward a decision...

You can make the following assumptions:

- The next PC logic will automatically increment the PC when you fetch a new instruction
- Both the sign-extended and zero-extended forms of the immediate field are available to Bus B
- The register specifiers Rs, Rt, and Rd are always correctly passed to the register file (data written into the register file is passed from BusB)
- Both the register file and the memory have purely combinational (asynchronous) reads
- The register file, memory, and all registers are triggered by the same clock edge (with no skew)



You can use the rest of this page for scratch space. The first question begins on the next page...

A) [10 points] Microprogram the following instructions by filling in the table. The **SrcA** and **SrcB** fields specify which signals will be assigned to BusA and BusB, respectively. The **WrDest** field specifies what component is written at the end of the cycle. This can be any one of the state registers (A and B can be paired together in one cycle), the register file, or memory. It is implied that all other components will not be written accidentally. The **Sequence** field behaves as presented in class: it specifies whether the microprogram should return to the fetch stage to start a new macroinstruction, dispatch to a location specific to the current opcode, or proceed in order. You should ignore the BNE instruction for now - it is only provided for part (C). *Hint: you do not need to fill all the rows.*

µAddr	**Instruction**	**SrcA**	**SrcB**	**ALUOp**	**WrDest**	**Sequence**
00	Fetch					
01	Decode					
02	ADD					
03						
04						
05						
06	ADDI					
07						
08						
09						
0A	LW					
0B						
0C						
0D						
0E	SW					
0F						
10						
11						
12	*BNE*	*A*	*B*	*Sub*	--	*If ALUZero Then Next Else Fetch*
13		*PC*	*SX*	*Add*	*PC**	*Fetch*

B) [10 points] One of the major drawbacks to using large buses is the massive loading caused by so many components all connected to the same node. To approximate the effect of the increased loading, assume that it takes an additional 10ns just to drive a signal on a bus (in other words, assume the hollow arrowheads in the datapath schematic have a delay of 10ns). Each component delay is copied below for your reference (you'll notice they all match the other problems). *Neglecting the BNE instruction,* ***what is the maximum clock frequency of this processor?***

Registers (clk-to-Q)	1ns
Register File	10ns
Extender	3ns
ALU	15ns
Memory	10ns
nPC Logic	10ns

C) [10 points] Calculate the execution time of the following assembly program, which adds two 1000-element integer vectors. The base pointers of the source vectors and the destination vector are initially stored in $4, $5, and $6 respectively. *Give your reasoning and/or equations in words, and* ***clearly*** *substitute each term with its actual numerical value!*

```
     ADDI $0   $8,1000
top: LW   $10, 0($4)
     LW   $11, 0($5)
     ADD  $12, $10,  $11
     SW   $12, 0($6)
     ADDI $4,  $4, 4
     ADDI $5,  $5, 4
     ADDI $6,  $6, 4
     ADDI $8,  $8, -1
     BNE  $8,  $0, top
```

Problem 3: MIPS 5 Stage Pipelined Processor (35 points)

The processor shown on the next page is one implementation of a standard MIPS 5 stage pipelined processor. The specifications of the processor are as follows:

- Five pipeline stages: IF, ID, EX, MEM, WB.
- The processor does not have forwarding nor hazard implemented yet.
- Control signal are represented by dotted lines. ***Control bus values represent the actual control bits top-down.*** Control signal values are in ***binary format.*** Data signal values are in ***decimal format.***
- RegisterFile ***writes in the first half of the cycle***, and ***reads in the second half.***
- Both instruction and data memory reads asynchronously; data memory writes synchronously. Both memories are ***byte addressed***. Data memory size is 16 words and only the least significant bits are used to address the memory.
  Each component has been labeled with a ***worst-case delay***. Registers only have clock-to-Q delay, and ***zero setup/hold time. Assume no clock skew***.
- The diagram of the processor also includes a snapshot of the processor ***at the beginning of the cycle***, with necessary signal values labeled.
- Tables 1 & 2 show the contents of the register file and data memory at the snapshot time. ***Assume the register file and memory have been correctly updated in the previous cycle, but all writes in the current cycle have not taken place yet.*** Table 3 shows the ALU control truth table. A MIPS R3000 opcode table is also attached at the end.

A) [5 points] How many branch delay slot(s) does this processor have?

B) [5 points] What is the minimum clock period of the processor?

C) [7 points] In MIPS assembly language, determine exactly what instruction is being executed in ID stage? What are the changes to the register file and data memory after the completion of the instruction?

D) [7 points] In MIPS assembly language, determine exactly what instruction is being executed in EX stage? What are the changes to the register file and data memory after the completion of the instruction?

E) [11 points] Now implement only the hazard detection unit without forwarding, such that from the software's perspective, the processor has exactly one branch delay slot, exactly one load delay slot, and all hazards are resolved at the hardware level. You are only allowed to add write enable control signals to any of the five pipeline registers, along with necessary control logic.

i. In order to stall the pipeline, some of the five pipeline registers should have write enable control signal. Fill in the following table. ('Y' means yes needed, `N' means not necessary)

	PC	IF/ID	ID/EX	EX/MEM	MEM/WB
Need Write En?					

ii. Describe in words, or pseudocode, under what condition each of the pipeline registers should be write disabled, and for ***how many processor cycles*** in each case.

Pipeline Register	Write Disable Condition	Number of Cycles
PC		
IF/ID		
ID/EX		
EX/MEM		
MEM/WB		

Table 1: Register File Contents

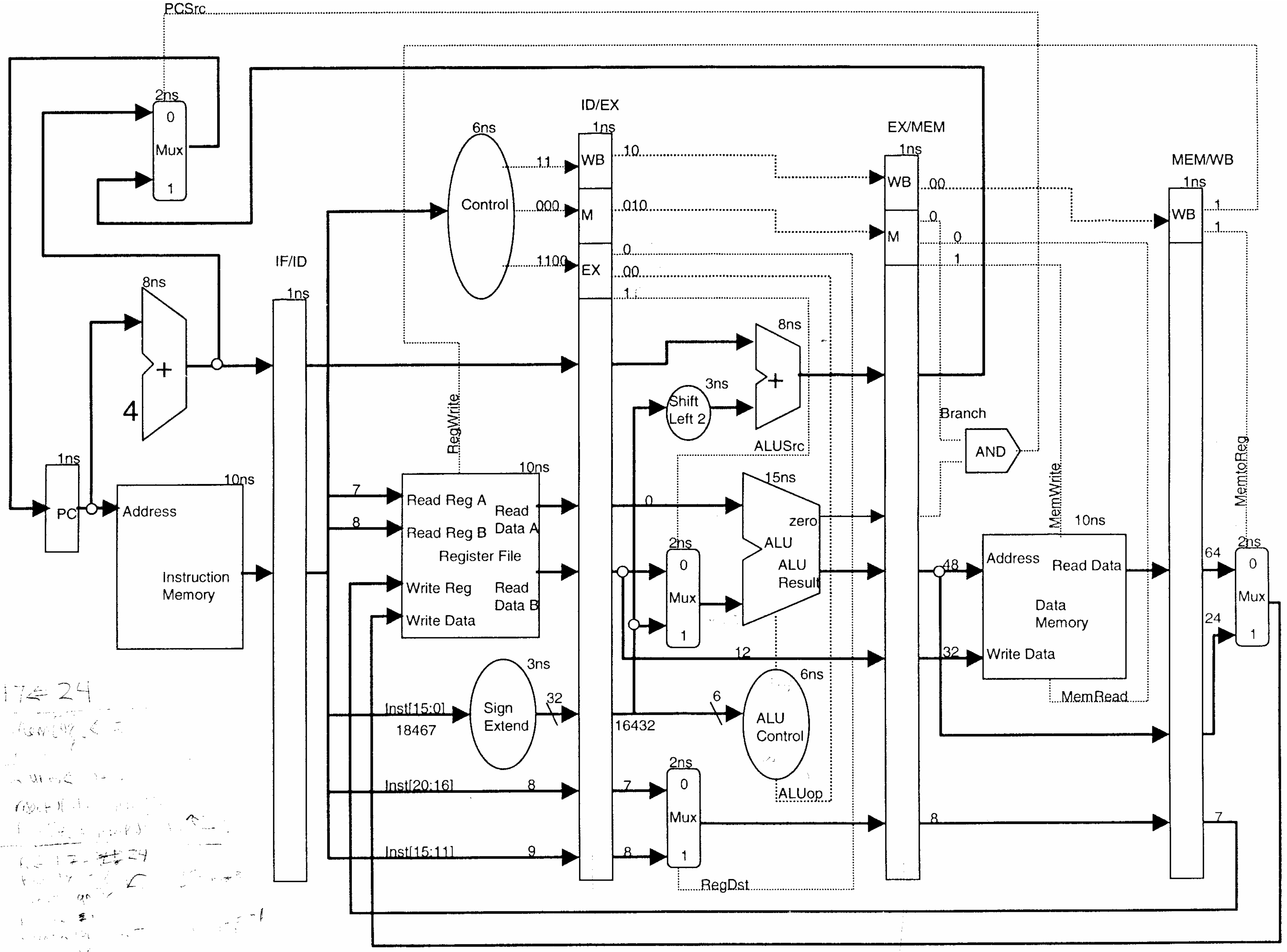
Register Address	Content
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10
11	11
12	12
13	13
14	14
15	15
16	16
17	17
18	18
19	19
20	20
21	21
22	22
23	23
24	24
25	25
26	26
27	27
28	28
29	29
30	30
31	31

Table 2: Data Memory Contents

Word Address	Content
0	24
1	78
2	46
3	184
4	169
5	23
6	78
7	40
8	80
9	36
A	85
B	42
C	56
D	72
E	81
F	100

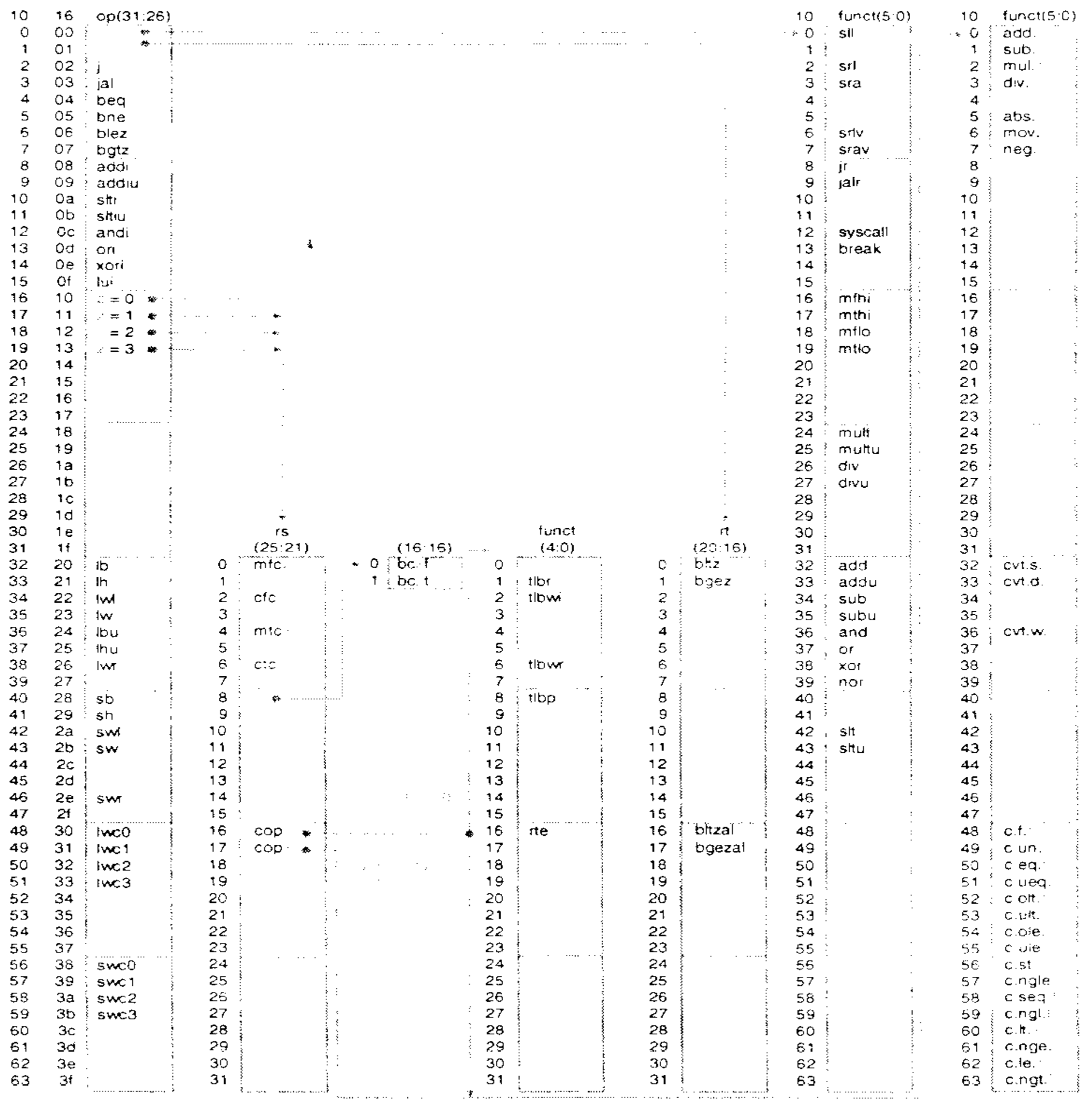
Table 3: ALU control truth table

ALU opcode	ALU operation
00	Add
01	Sub
10	R-format


PCSrc
2ns
0
Mux
1
8ns
+
4
1ns
PC
10ns
Address
Instruction Memory
IF/ID
1ns
6ns
Control
11
000
1100
RegWrite
7
8
10ns
Read Reg A
Read Reg B
Register File
Write Reg
Write Data
Read Data A
Read Data B
3ns
Sign Extend
Inst[15:0]
18467
32
Inst[20:16]
8
Inst[15:11]
9
ID/EX
1ns
WB
M
EX
10
010
0
00
1
16432
7
8
3ns
Shift Left 2
8ns
+
ALUSrc
15ns
zero
ALU
ALU Result
2ns
0
Mux
1
12
6
6ns
ALU Control
ALUop
2ns
0
Mux
1
RegDst
EX/MEM
1ns
WB
M
00
0
0
1
Branch
AND
MemWrite
10ns
Address
Read Data
Data Memory
Write Data
48
32
MemRead
8
MEM/WB
1ns
WB
1
1
MemtoReg
64
24
7
2ns
0
Mux
1

17 = 24

[illegible]



10	16	op(31:26)
0	00	*
1	01	*
2	02	j
3	03	jal
4	04	beq
5	05	bne
6	06	blez
7	07	bgtz
8	08	addi
9	09	addiu
10	0a	slti
11	0b	sltiu
12	0c	andi
13	0d	ori
14	0e	xori
15	0f	lui
16	10	z = 0 *
17	11	z = 1 *
18	12	z = 2 *
19	13	z = 3 *
20	14	
21	15	
22	16	
23	17	
24	18	
25	19	
26	1a	
27	1b	
28	1c	
29	1d	
30	1e	
31	1f	
32	20	lb
33	21	lh
34	22	lwl
35	23	lw
36	24	lbu
37	25	lhu
38	26	lwr
39	27	
40	28	sb
41	29	sh
42	2a	swl
43	2b	sw
44	2c	
45	2d	
46	2e	swr
47	2f	
48	30	lwc0
49	31	lwc1
50	32	lwc2
51	33	lwc3
52	34	
53	35	
54	36	
55	37	
56	38	swc0
57	39	swc1
58	3a	swc2
59	3b	swc3
60	3c	
61	3d	
62	3e	
63	3f	

	rs (25:21)
0	mfc
1	
2	cfc
3	
4	mtc
5	
6	ctc
7	
8	*
9	
10	
11	
12	
13	
14	
15	
16	cop *
17	cop *
18	
19	
20	
21	
22	
23	
24	
25	
26	
27	
28	
29	
30	
31	

	(16:16)
0	bc.f
1	bc.t

	funct (4:0)
0	
1	tlbr
2	tlbwi
3	
4	
5	
6	tlbwr
7	
8	tlbp
9	
10	
11	
12	
13	
14	
15	
16	rte
17	
18	
19	
20	
21	
22	
23	
24	
25	
26	
27	
28	
29	
30	
31	

	rt (20:16)
0	bltz
1	bgez
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	bltzal
17	bgezal
18	
19	
20	
21	
22	
23	
24	
25	
26	
27	
28	
29	
30	
31	

10	funct(5:0)	10	funct(5:0)
0	sll	0	add.
1		1	sub.
2	srl	2	mul.
3	sra	3	div.
4		4	
5		5	abs.
6	srlv	6	mov.
7	srav	7	neg.
8	jr	8	
9	jalr	9	
10		10	
11		11	
12	syscall	12	
13	break	13	
14		14	
15		15	
16	mfhi	16	
17	mthi	17	
18	mflo	18	
19	mtlo	19	
20		20	
21		21	
22		22	
23		23	
24	mult	24	
25	multu	25	
26	div	26	
27	divu	27	
28		28	
29		29	
30		30	
31		31	
32	add	32	cvt.s.
33	addu	33	cvt.d.
34	sub	34	
35	subu	35	
36	and	36	cvt.w.
37	or	37	
38	xor	38	
39	nor	39	
40		40	
41		41	
42	slt	42	
43	sltu	43	
44		44	
45		45	
46		46	
47		47	
48		48	c.f.
49		49	c.un.
50		50	c.eq.
51		51	c.ueq.
52		52	c.olt.
53		53	c.ult.
54		54	c.ole.
55		55	c.ule
56		56	c.st
57		57	c.ngle
58		58	c.seq
59		59	c.ngl.
60		60	c.lt.
61		61	c.nge.
62		62	c.le.
63		63	c.ngt.