University of California, Berkeley
College of Engineering
Department of Electrical Engineering and Computer Science

CS152 Midterm Solution

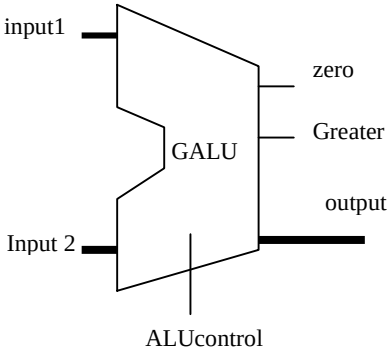
Prof. Bob Brodersen

Fall, 2000

Problem 1: Single-cycle Processor and Performance Evaluation

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 $s1 $s2; cslw $t $s1 $s2; cslwcs $t $s1 $s2 .

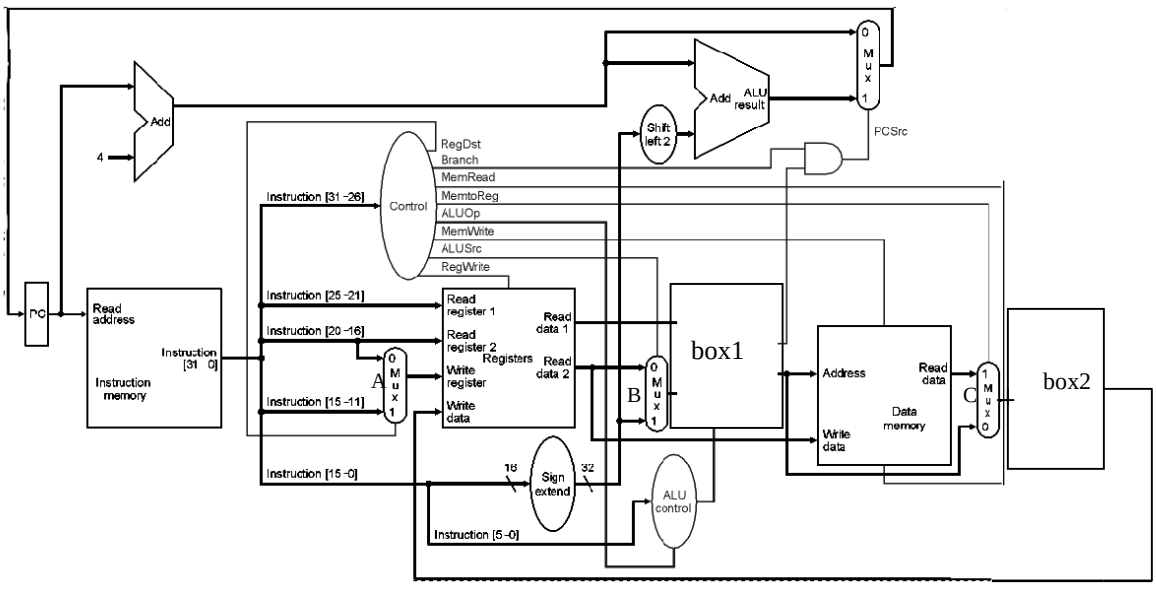


Besides the usual PC handling and instruction fetch as before,

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

So now you see that cs and lw in above names stand for compare_select and loadword.

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 he can use 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.

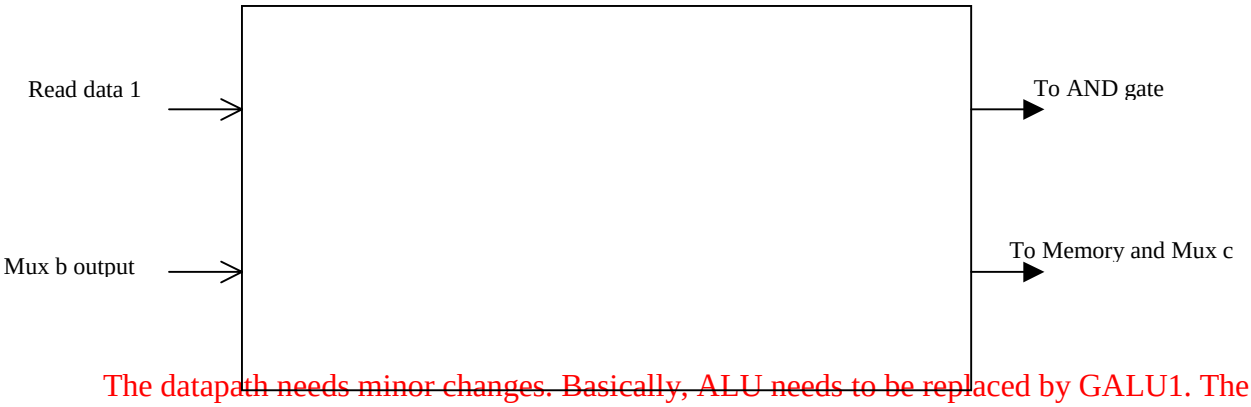


Questions:
a). **Describe below in words and fill in box 1** the blocks needs to be added in order to implement the cs and cslw instruction. **Furthermore add in and draw in box 2** any additional blocks to implement the cslwcs instruction. (hint: think about doing so by adding GALU(s), MUX(es),AND(s) etc., and possibly new control signal(s))

i) For cs and cslw:

The datapath needs minor changes. Basically, needs a GALU1. The Greater1 signal is used to select between $s1 and $s2 (so a mux is needed), and another control signal NEWC1 is used to choose between this output and the GALU data output (another mux is needed). In total 1 GALU and 2 mux needed

Box 1



The datapath needs minor changes. Basically, ALU needs to be replaced by GALU1. The Greater1 signal is used to select between $s1 and $s2 (so a mux is needed), and another control signal NEWC1 is used to choose between this output and the GALU data output (another mux is needed). In total 1 GALU and 2 mux needed

ii) For cslwcs:

One way to add cslwcs is add another generalized ALU2 after the right most mux to compare the loaded word and $s1 (subtraction and generate GREATER2), and then add a new mux with control signal NewC2 such that NewC2&Greater2 selects the value between $s1 and memory output. In general 1GALU, 1 mux and 1 AND are needed.

Box 2

b). Fill in the control table below with X, 0 or 1. Add in new column(s) if you need new control signal(s) (you may not need all three columns)

Instruction	Alusrc	Memto Reg	Reg Write	Mem Read	Mem Write	NewC1	NewC2	
Sw	1	X	0	0	1	0	0	
R-format	0	0	1	0	0	0	0	
cslw	0	1	1	1	0	1	0	
cslwcs	0	1	1	1	0	1	1	

c). Please point out the **critical path**, estimate for him the **critical path delay in ns**, and hence determine the **fastest clock rate in MHz**. Finally use a program given below to **evaluate the processor**. Use the following delay parameters (ignore hold time and regard clock-to-Q time as the delay, consider setup time for register files write delay):

- 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) Critical path is (write down the components' names in order):

See below

ii) Critical path delay is (ns) ? and fastest clock rate in Mhz ?

The original processor has delay = loadword delay = PC+ instruction mem + register file + mux + ALU+data memory + mux+ register write =1+10+10+2+15+10+2+10=60 ns

Now have additional 2 mux + 1GALU+1mux (ignore and delay) = 51+ 4+ 15+2=81 ns

The fastest CLK Rate= 1/81ns= 1000/81 MHz

(note that if you used different way in part a), your result here will be different as well, which is OK)

iii) Now use a program to evaluate the performance of this new 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?

With this new processor, the time = 1,000,000 * 81ns = 81ms

d). **Please write down an as-simple-as-possible decomposition** of the cslwcs instruction in terms of cs,cslw, lw, sw and/or R-format instructions. Thus the original processor can be modified only to run additional cs and cslw in one cycle and use multi-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.

i) cslwcs $t $s1 $s2 ⇔ (in terms of cs, cslw, lw, sw and R-format)

cslw $temp $s1 $s2
cs $t $s1 $temp

ii) This time worst case delay =

60+ 2mux=64ns,

iii) so the new processor need to use ?(ms) to run the program

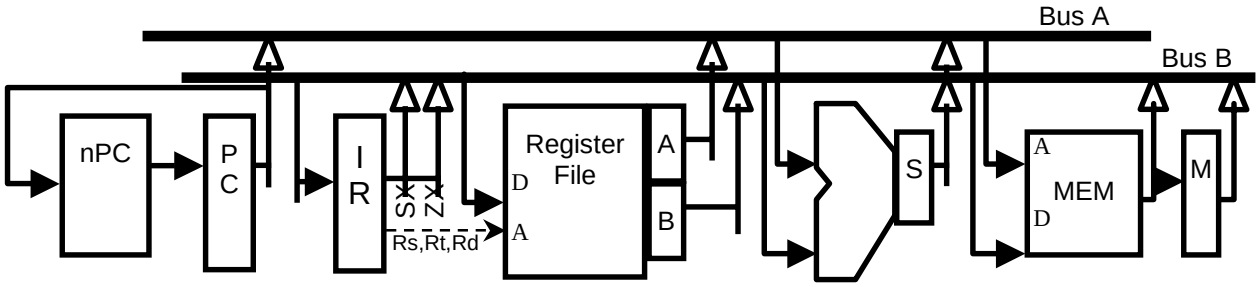
(1,000,000*10%+1,000,000)*64 = 70.4 ms,

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 BEQ 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	PC	Mem	–	IR	Next
01	Decode	–	–	–	A,B	Dispatch
02	ADD	A	B	Add	S	Next
03		–	S	–	RegFile	Fetch
04						
05						
06	ADDI	A	SX	Add	S	Next
07		–	S	–	RegFile	Fetch
08						
09						
0A	LW	A	SX	Add	S	Next
0B		S	–	–	M	Next
0C		–	M	–	RegFile	Fetch
0D						
0E	SW	A	SX	Add	S	Next
0F		S	B	–	Mem	Fetch
10						
11						
12	*BNE*	*A*	*B*	*Sub*	–	*If ALUzero Then Next Else Fetch*
13		*PC*	*SX*	*Add*	*PC**	*Fetch*

Note that for the LW instruction, given the way the problem was defined it was acceptable to have the last two cycles merged into one. Therefore, if your answer had the memory access and write back both occur in the fourth cycle, you should not have lost points.

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

The critical path of this processor is during instruction fetch, where the buses must be accessed twice, along with the memory and register delay. The minimum cycle *time* is therefore

Time = clk-to-Q + bus_access + memory + bus_access
= 1 + 10 + 10 + 10
= 31 ns

The maximum clock frequency is the inverse of time, such that

Freq = 1/Time
= 1/31 ns
≈ ***32.258 MHz***

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
```

Based on our microprogramming above, we get the following CPI for each instruction: ADD = 4, ADDI = 4, LW = 5, SW = 4, and BNE = 3 *or* 4. The trick is noticing that the BNE instruction takes a different number of cycles based on the branch condition.

Since this is a finite closed loop, we know that exactly 999 branches are taken, and the last one is not taken. That gives us the following instruction counts: ADD = 1000, ADDI = 4(1000)+1, LW = 2(1000), SW = 1000, BNE_{taken} = 1, BNE_{not_taken} = 999. We now can find the number of cycles needed to execute the program by summing the products of CPI and instruction count:

Cycles = Σ(CPI * #Instructions)
= 4(1000) + 4(4001) + 5(2000) + 4(1000) + 3(1) + 4(999)
= 4000 + 16004 + 10000 + 4000 + 3 + 3996
= 38,003

Execution time is now just the product of cycle count and cycle time:

Time = Cycles * CycleTime
= 38,003(31 ns) = ***1.178 ms***

Problem 4: MIPS 5 Stage Pipelined Processor (30 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 following:

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 value represents the actual control bits from top down.*** Control signal values are in ***binary format***. Data signal values are in ***decimal value***.

RegisterFile ***writes in the first half of the cycle***, and ***reads in the second half***

Both instruction and data memory read 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 ***worst-case delay***. Registers only have clock-to-Q delay, ***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.

Table 1 & 2 shows the contents of the Register file, Data memory at the snapshot time. ***Assume 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. MIPS R3000 Opcode table is also attached at the end.

A) How many branch delay slot(s) does this processor have? (4 points)

3

B) What is the minimum clock period of the processor? (4 points)

1+6+15 = 22 ns

C) In MIPS assembly language, determine exactly what instruction is being executed in ID stage? What's the result? *i.e., register file or data memory content changes. (6 points)*

Sub $9, $7, $8
$9 = 24 – 8 = 16 Note: WB stage write to $7 with value 24

D) In MIPS assembly language, determine exactly what instruction is being executed in EX stage? What's the result? *i.e., register file or data memory content changes. (6 points)*

Lw $7, 16432($0) Note: it's the same word address as 48($0)
$7 = Mem(48/4) = 32 Note: MEM stage SW to address 48 with value 32

E) 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 should be resolved on hardware level. You are only allowed to add write enable control signals to any of the five pipeline registers, along with necessary control logics. *(10 points)*

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

	PC	IF/ID	ID/EX	EX/MEM	MEM/WB
Need Write En?	**Y**	Y	N	N	N

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

Pipeline Register	Write Disable Condition
PC	If ID detects BCH, disable for 2 cycles; If ID detects LW, disable for 1 cycles; If EX instruction writes back to the same register as ID instruction uses, disable for 2 cycles; If MEM instruction writes back to the same register as ID instruction uses, disable for 1 cycles;
IF/ID	Same as PC disable conditions
ID/EX	Never
EX/MEM	Never
MEM/WB	Never

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 Content

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