SOLUTIONS

EE457: Computer Systems Organization
Summer 2020 - Midterm Exam
06/23/20, 1:30PM – 3:30PM (Submit by 4:30 p.m)

Name: ___Solutions_____

Student ID: _____

Email: _____@usc.edu

Lecture section:

Redekopp		
1:30 p.m.		

[5 points: Complete all the information in the box above.]

Page	Ques.	Your score	Max score	Recommended Time
1	-			0 min.
2	1		14	15 min.
3-5	2		36	45 min.
6	3		15	15 min.
5	4		8	15 min.
6	5		6	15 min.
7	6		9	15 min.
	Total		88+2=90	

Note: The last page is blank and can be used for scratch paper.

~~Please turn it in with your exam~~ (for in person exams only)

SOLUTIONS

1. **(15 pts.) Short Answer** [Fill in the blank at the start of the question with the appropriate answer to the question or the selection that makes the statement true.]

   a. _**LATER**____ Even with full forwarding in a pipelined processor, a stall may be required if the stage that produces the result is (**later / earlier**) in the pipeline than the stage where the dependent instruction consumes the result.

   b. _ **3** ___ A state in a state diagram has 2 outgoing transitions and 3 incoming transitions. If using a 1-hot implementation, the flip-flop for that state in the NSL will be produced by an OR gate with how many inputs?

   c. __ **STATIC** ___ (**Dynamic / Static**) instruction count will determine how much memory the program occupies when executing.

   d. __**FALSE**___ (**True / False**) Suppose half of the instructions executed by a program are ADD instructions which take 2 cycles. Reducing the ADD instructions by half will cut the overall execution time in half.

   e. __**FALSE**___ (**True / False**) In the single-cycle CPU, reducing the latency of the datapath required to execute an **JUMP** instruction is likely to decrease the clock period.

   f. __**TRUE**__ (**True / False**) In the single-cycle CPU, reducing the latency of the datapath required to execute an **LW** instruction is likely to decrease the clock period.

   g. ___**3**_____ What is the minimum number of gate delays required to implement a 6-input OR function using only 2-input OR gates?

   h. __**Moore**____ Signals that must be valid <u>during</u> the clock are best produced using (**Mealy / Moore**) style.

   i. __**False**__ **(True / False)** For signed numbers, the comparison of A < B can be performed by subtracting and then simply checking if the MSB of the result is a 1.

   j. Perform the indicated arithmetic operations, showing your work, for the specified representation system. Answers are limited to 8-bits (not 9-bits). (Do not use the borrow method for subtraction, use the 2's complement method of subtraction.) Finally, state whether overflow has occurred and **<u>briefly explain why or why not</u>**.

a.) 2's comp. system

$$\begin{array}{rr} & 11010101_2 \\ - & 01011110_2 \\ \hline & 11010101_2 \\ & 10100001_2 \\ + & \quad 1_2 \\ \hline & 01110111_2 \end{array}$$

Overflow: **<u>Y</u> / N**

b.) Unsigned system

$$\begin{array}{rr} & 01100101_2 \\ + & 01010100_2 \\ \hline & 10111001_2 \end{array}$$

Overflow: **Y / <u>N</u>**

SOLUTIONS

2. **(36 pts.) Arithmetic, Datapath, and Control Unit Design**. Recall that sign-truncation refers to removing copies of the sign bit without altering the number's value. For example, **11111010** binary can be truncated by **4 bits** and reduced to **1010** without affecting the value.

   Given an array of 32 (8-bit) **signed (2's complement) binary numbers** stored in a memory, scan through the array, and record the MAX number of bits that can be truncated from any number in the array (i.e. find the number with the most copies of the sign-bit and record how many copies that number had.) In addition, by the end of the scan, set a FLAG bit if ANY number in the array had **4 or more** leading sign bits that could be truncated.

   Below is a sample of 4 (though you will examine 32) numbers and how the MAX and FLAG variables would update based on the computation of each number. The final result would be MAX = 6 and FLAG = 1.

Memory Contents	*Addr. (i)*	0	1	2	3
	M[i]	1**1**011101	0**0000010**	1**111111**0	10110111
S = (Sign-bits that can be Truncated from M[i]		1 (remember you must retain the original sign bit)	5	6	0
MAX , FLAG		1 , 0	5 , 1	6 , 1	6 , 1

   While there are many ways to accomplish this computation, we want you to scan through each number and examine the bits of the current number 1 clock cycle at a time (see the datapath below). While we could go through all 8-bits, since we only care about how many leading sign bits the number has we should not require 8 cycles per number if there are fewer leading sign bits. For example, given the number **11100101**, we should be able to **stop counting the sign bits after 2 clock cycles** since there are only 2 leading copies of the sign bit. This is a performance **requirement**. **So if a number has n leading copies of the sign bit, you may only use n cycles to count the sign bits.**

   To produce the answers, we will keep two counters: the **i-counter** to track which array element we are examining and the **B-counter** to select (via a mux) one of the bits of the number per cycle. For this B-counter you will need to complete its connections with the ADDER/SUBTRACTOR and mux to ensure the appropriate select bit sequence is generated. We will use a third counter (**S-counter**) to count how many leading copies of the sign bit there are. Finally, we will use a register to store the **MAX** number of leading sign bits from ANY number and a separate 1-bit register for the **FLAG** to indicate if ANY number had **4 or more copies** of the sign bit.

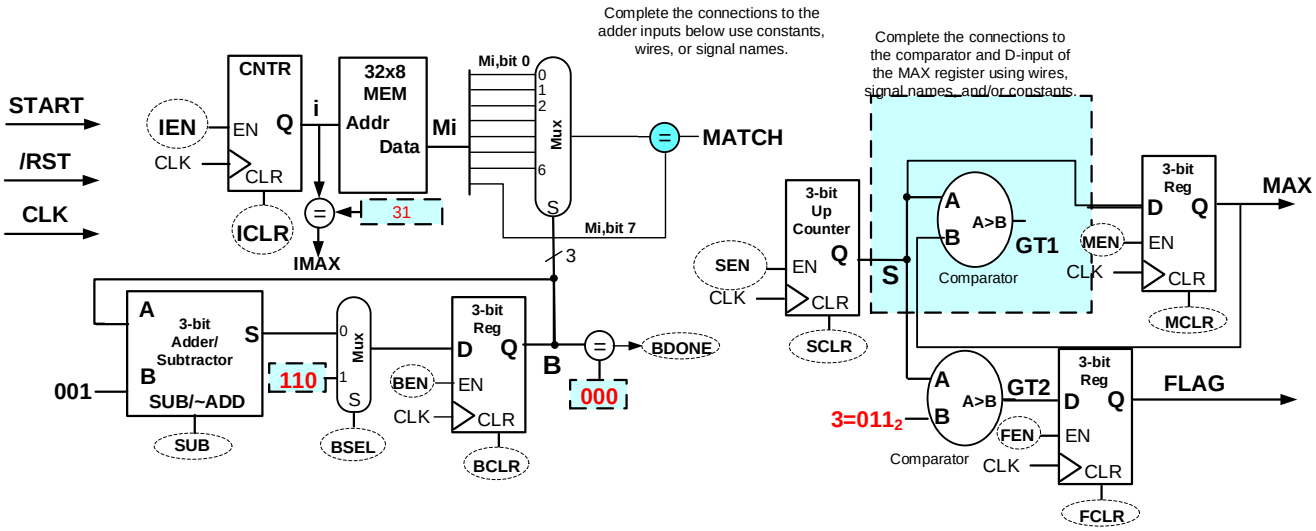
   We will use a Mealy-style state machine. After an initial state (**INIT**) we will move to the BCNT (bit counting) state where we will count the number of leading sign bits in the current array element. Once we determine that value, we will enter a SCMP (S compare) state to update the MAX and FLAG outputs appropriately. We can then return to the BCNT state to process the next number and repeat this process until we are done.

   Examine the incomplete datapath on the next page and the skeleton of the state machine on the following page. Complete the state machine and then the data path by filling in the shaded regions, taking care to read the instructions for what is expected. Finally, complete the NSL and OFL.

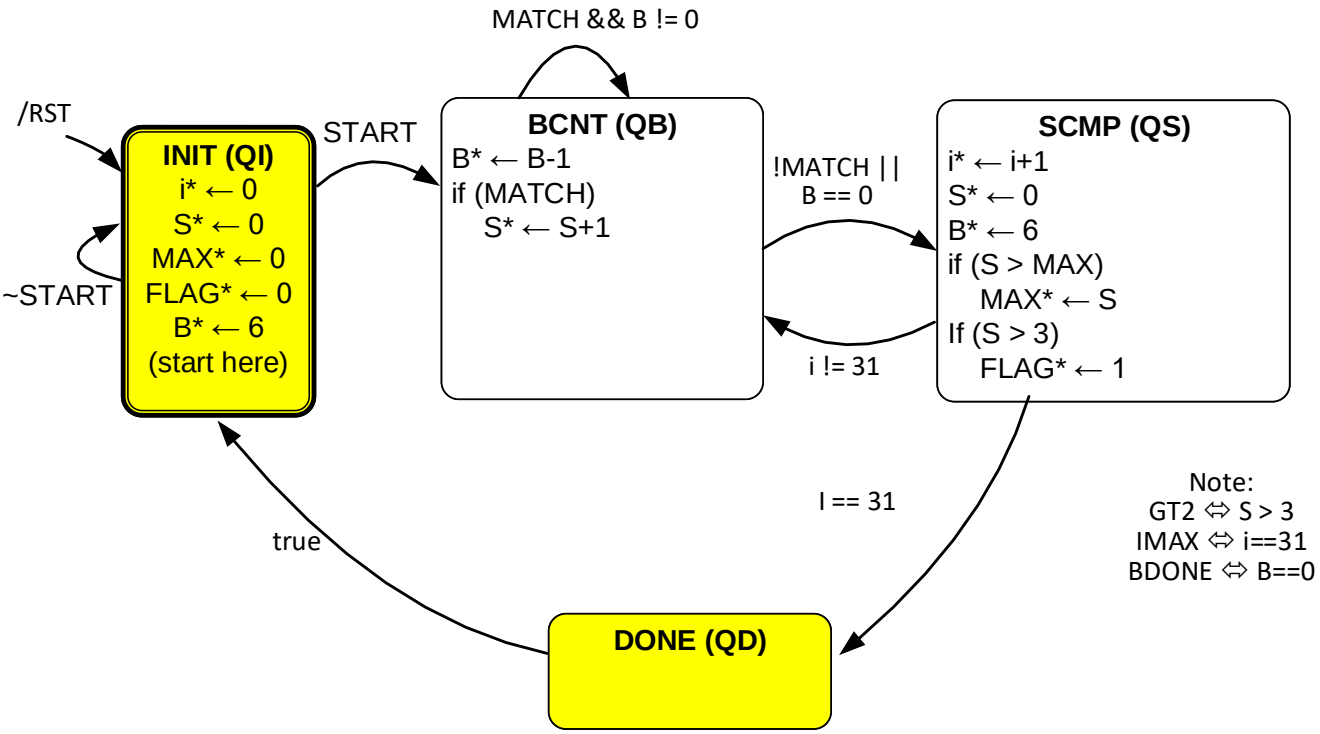
SOLUTIONS

a.) [1 pt] How many bits should the i-counter be? ___**5**_____

b.) [7 pts] Complete the datapath connections below by using constants, drawing wires, or simply using signal name labels to the inputs in the shaded boxes. The signals in the dotted circles will be generated by the OFL of the state machine or other signals from the datapath below. Those will be dealt with on the next page.



c.) [12 pts] Complete both transitions & operations that should happen in the **BCNT** and **SCMP** states.



SOLUTIONS

(7 pts.)

e.) Now suppose we implemented the state machine from the previous part and gave you the one-hot state outputs: QI, QB, QS, and QD. Use these signals plus **others that are defined in the datapath** to define the control signals indicated below: Just write logic equations using logic operations

MEN = __QS * GT1_____

SEN = ___QB * MATCH_____

BSEL = ___QI + QS _____

BEN = ____ QI + QS + QB // or 1_____

IEN = _____QS (+QI is not needed but ok)_____

SCLR = ____QI+QS_____

In the datapath, the 1-bit comparator producing MATCH can be implemented with a single logic gate. What type of gate would be correct: __XNOR_ (AND / XOR / XNOR / NOR)

(6 pts.)

f.) Below is a **subset** of the state flip-flops. Assuming a **1-hot state assignment**, complete the NEXT STATE LOGIC equations and SET/CLR inputs of the state memory. You need not draw the gates but just write the equations for the specified D inputs. Then in the shaded boxes show what to connect to the **SET/CLR inputs** (Vdd and GND are always available)

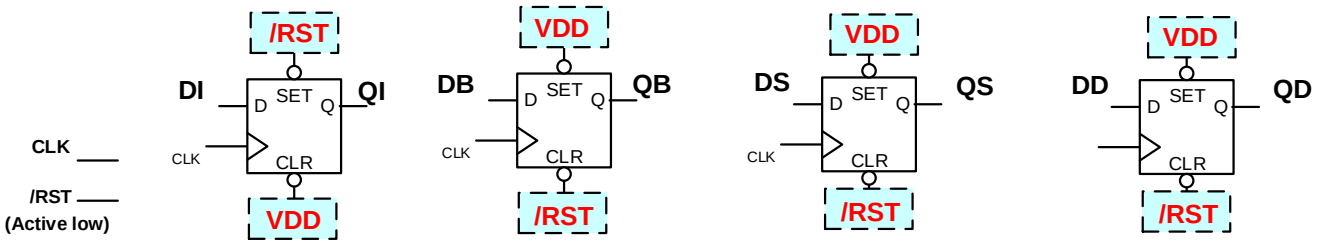
DI = _QI*~START + QD_____

DB = ___QI*START + QB*MATCH*B!=0 + QS*(I != 31) ____
Note: IMAX⇔i==31, GT2 ⇔ S>3_____

DS = ___QB*(!MATCH + B==0)_____Note: BDONE ⇔ B==0_____

(3 pts.)

g.) Using /RST, constants and other signals, show how to connect the /SET and /CLR inputs of the following state flip-flops.



SOLUTIONS

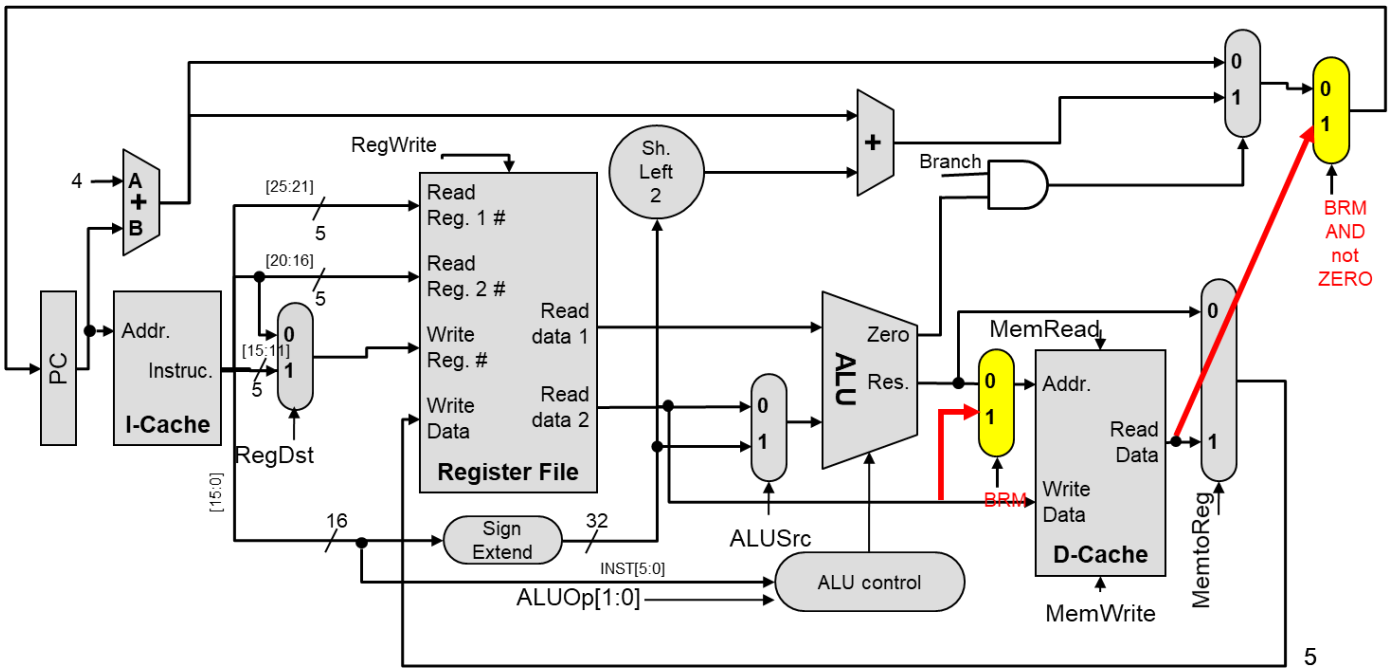
3. **(15 pts.) ISA and Single-Cycle CPU Datapath**: We want to add support for a new instruction '**BMNE**' (**Branch Memory Address if Not Equal**), while not affecting any other instructions. This new instruction has the format shown below and will branch to the value loaded from memory given by the address in $rt if the value in $rs is not equal to a sign-extended immediate (i.e. `$rs != imm`), otherwise it continues executing sequentially (the next instruction). Implement any changes to the datapath and control signals to support this new **BMNE** instruction on the single-cycle CPU. Assume when this instruction executes a new **BMNE** control signal will be generated and set to '1'.

```
BMNE $rs,($rt),imm  instruction description:
if($rs != imm)
   PC = M[$rt]
```

This instruction will use the machine code I-format:

BMNE:

opcode	rs	rt	16-bit signed immediate
6	5	5	(immediate to be compared to $rs)



a. What operation should the ALU perform? _____SUBTRACT_____

b. Sketch the additions/changes to the **datapath** above that would be needed to support this new instruction and its operations (provide a brief description in the space below if the sketch is unclear). Use as little additional logic as possible.

See diagram above

c. Show the values of the following control signals to implement this new instruction.

	BMNE	MemToReg	Branch	ALUSrc	RegWrite	MemRead	MemWrite
Value (0,1,X)	1	X	X	1	0	1	0

SOLUTIONS

4. (8 pts.) Performance

Assume a program requires the execution of the following number of instructions with the given CPI. The processor uses a 2 GHz clock period.

Instruction Type	Instruction Count	CPI
ADD	$100*10^6$ instructions	1
Branch	$50*10^6$ instructions	2
Load	$50*10^6$ instructions	3
Store	$25*10^6$ instructions	3
MUL	$25*10^6$ instructions	5

a.) What is the execution time of this program (in milliseconds)?

Time: __275_____ ms *(show work)*

$(100*1 + 50*2 + 50*3 + 25*3 + 25*5) * 10^6 = 550*10^6$ total clks / $2*10^9$ Clks/sec =275 ms

b.) Is it possible to achieve a 1.1x speedup by improving the CPI of only the BRANCH instructions?

Yes / No: *(show work)*

Orig. clocks = 550E6 clocks
New clocks (Branch CPI reduced to 1 (ideal) will lead to 50E6 less clocks) = 500 E6 clocks
Speedup = 550E6 / 500E6 = 1.1x

c.) If we want the program to run 1.2x faster by simply reducing the **number of Load** instructions, by what **factor** would we need to reduce the instruction count of Loads?

Factor of reduction (i.e. old # of LOADS / new # of LOADS): __18/7 = 2.57_____ *(show work)*

Loads account for 3*50E6 of the 550E6 clocks (time) of the original program so we can use Amdahl's law:
Speedup = 1.2 = 1 / (400/550 + 150/550/x).
x = 18/7 = 2.57

d.) Assume each MUL instruction is converted to 4 ADD instructions. Assuming we execute that updated program on the pipelined processor discussed in class with full-forwarding, 1 cycle stall for Loads followed by dependent instructions and 1 cycle branch penalty (i.e. early-branch determination), what would the runtime of the program be in the worst case?

Time: __212.5_____ ms *(show work)*

WE will now have 100E6 more ADD instructions replacing the 25E6 MUL instructions. So with the pipelining the ideal CPI is 1 (1 cycle per instruction). However the 50E6 LW will incur an extra 50E6 cycles in the worst case (assuming they are all followed by dependent instructions) and the 50E6 branches will incur a 1 cycle branch penalty. Thus the time will be: (200E6*1 + 50E6*2 + 50E6*2 + 25E6*1) / 2E9 = 212.5 ms

SOLUTIONS

5. (5 pts) Pipelining Performance

Consider the final pipelined CPU implementation we arrived at in class with **full forwarding** where possible, **stalling logic** for a LW followed by a dependent instruction, and branch determination in the DECODE stage (**i.e. early branch determination**). Look at the code segment below which implements a loop that starts at L1 and will exit when the 'bne' instruction is true but continue looping when 'bne' is false and thus the 'beq' will jump back to the top of the loop. I

a.) Suppose the bne is false twice and then is true on the 3rd iteration of the loop. How many total bubbles (stall cycles + flushed instructions) would occur during execution of those 3 iterations?

b.) Total _**6**_ = _**2**_(1 for LW 1 for taken BEQ) + _**2**_ + _2 **(for LW and now BNE is taken)**_
(1st iteration) (2nd iteration) (3rd iteration)

Original Code Segment 1

`L1: add  $4, $8, $9`
`    lw   $3, 20($4)`
`    sub  $6, $3, $5`
`    addi $4, $4, 8`
`    bne  $4, $9, L2`
`    sw   $8, 0($4)`
`    beq  $0, $0, L1`
`L2: or   $7, $6, $9`

Now assume the hardware designer REMOVES the hazard detection unit and flushing logic (forwarding logic is still present) and instead declares **1 delay slot after a LW** followed by a dependent instruction and **1 delay slot after a branch**. Answer the questions below about how to reorder the code as necessary to **ensure correctness** while also achieving the **BEST POSSIBLE performance** (i.e. CPI). **Recall that 'nop' instructions are always available for use to ensure correctness.**

c.) What instruction could be moved into the delay slot for the **bne** (answer 'nop' if no other instruction can be moved)?

___sub__(all others are dependent)_____

d.) What instruction could be moved into the delay slot for the **beq** (answer 'nop' if no other instruction can be moved)?

_____sw__(all others are dependent)_____

6. **(9 pts.) Pipelining and Arithmetic**

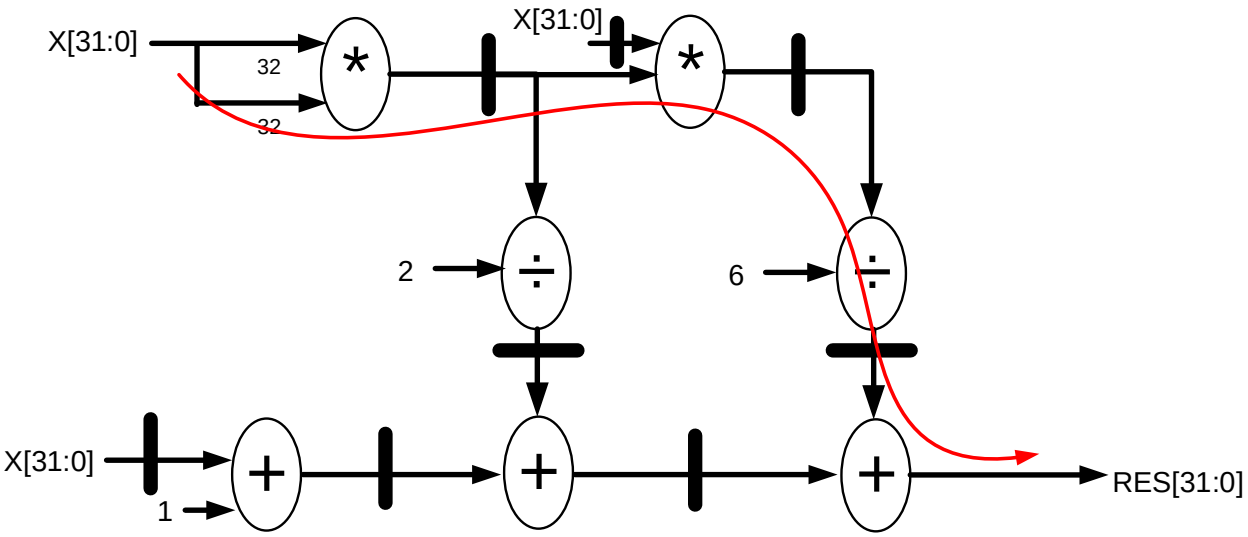
Consider the problem of calculating an approximation of e^x using the following formula:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!}$$

Given the datapath below assume the following delays:

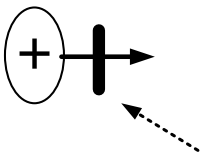
Addition unit	200 ps
Multiplication unit	500 ps
Division unit	800 ps

a. What is the total delay of the circuit as shown below? _**2000ps = (500+500+800+200)**.



b. Now suppose we have an array of many values for X (e.g. X[1000]) and we want to compute $e^{X[i]}$ for each value of X[i]. Go back to the circuit above and consider how we can pipeline the design. We want to achieve a **clock period of at most 800 ps** (i.e. the divider delay) with **a latency of 4 clock cycles and a throughput of 1 result every clock cycle in the steady state**. Show where to add pipeline stage registers by drawing a **THICK HORIZONTAL LINE** (See the example diagram below for how to draw your pipeline registers) through the wires were a pipeline register should be inserted.

Sample for how to draw a pipeline register on the design above:



A pipeline register at the output of the adder

SOLUTIONS

Intentionally blank for scratch work. Please turn it in with your exam:
Name: _____ Section time: _____