University of California, Berkeley – College of Engineering

Department of Electrical Engineering and Computer Sciences

Spring 2007 Instructor: Dan Garcia 2007-05-12

☹ CS61C Final Exam ☺

After the exam, indicate on the line above where you fall in the emotion spectrum between "sad" & "smiley"...

Last Name	Answers
First Name	
Student ID Number	
Login	cs61c-
Login First Letter (please circle)	a b c d e f g h i j
Login Second Letter (please circle)	a b c d e f g h i j k l m n o p q r s t u v w x y z
*Your **LAB** TA's name (please circle)*	Aaron / Alex / Brian / David / Matt / Michael / Valerie
Name of the person to your Left	
Name of the person to your Right	
All the work is my own. I have no prior knowledge of the exam contents nor will I share the contents with others in CS61C who have not taken it yet. ***(please sign)***	

Instructions (Read Me!)

- This booklet contains 9 numbered pages including the cover page. Put all answers on these pages (feel free to use the back of any page for scratch work); don't hand in any stray pieces of paper.
- Please **turn off** all pagers, cell phones & beepers. Remove all hats & headphones. Place your backpacks, laptops and jackets at the front. Sit in *every other* seat. Nothing may be placed in the "no fly zone" spare seat/desk between students. The exam is closed book, no computers, PDAs or calculators.
- Fill in the front of this page and put your name & login on every sheet of paper.
- You may use 2 pages (US Letter, front and back) of notes, plus the green COD 3/e reference sheet.
- There may be partial credit for incomplete answers; write as much of the solution as you can. We will deduct points if your solution is far more complicated than necessary. When we provide a blank, please fit your answer within the space provided. "IEC format" refers to the mebi, tebi, etc prefixes.
- **You must complete ALL THE QUESTIONS**, **regardless of your score on the midterm**. Clobbering only works from the Final to the Midterm, not vice versa. You have 3 hours...relax.

Problem	**M1**	**M2**	**M3**	**Ms**	**F1**	**F2**	**F3**	**F4**	**F5**	**Fs**	**Total**
Minutes	20	20	20	60	24	24	24	24	24	120	180
Points	**10**	**10**	**10**	**30**	**18**	**18**	**18**	**18**	**18**	**90**	**120**
Score	10	10	10	30	18	18	18	18	19	90	120

Midterm Revisited

M1) Do you remember when… we used to sing, sha la la la…

a) Some say bubble-gum pop music is garbage, so let's start with a question about memory management and running programs! BRIEFLY tell us why the first thing is better than the second thing:

Mark & sweep is better than **reference counting** because it…	can handle circular structures
Reference counting is better than **copying** because it…	can use all of memory
Copying is better than **mark & sweep** because it…	defragments memory so discontiguous free blocks can be re-absorbed into a whole
The **buddy scheme** is better than the **slab allocator** because it…	is adaptive to the requests coming in
The slab allocator is better than the **plain K&R free list** because it…	can handle small requests fast (eliminating the "pebbles")
Compiling is better than **interpreting** because…	executables are faster and you don't have to give away source code
Interpreting is better than **compiling** because…	the compile-link-run development process is slow and code is portable.
Dynamic linking is better than **Static linking** because…	smaller executables, library updates easy, mem footprint small (link as needed)
Static linking is better than **Dynamic linking** because…	of faster runtime performance, no worry about missing DLLs

b) Decode the binary numbers into MIPS instructions *with proper register names* (`$s0`, `$t0`, etc.). If there are any memory addresses, represent them in hex.

Address	32-bit Binary Instruction	Type (R, I, J)	MIPS Instruction w/args
0xAFFFFFF8	0000 0001 0000 1000 0100 0000 0010 0110	R	xor $t0, $t0, $t0
0xAFFFFFFC	0001 0100 0000 1000 1111 1111 1111 1110	I	bne $0, $t0, -2
0xB0000000	0000 1000 0000 0000 0000 0000 0000 0001	J	j 0xB0000004
0xB0000004	...whatever...	...whatever...	ori $v0, $0, 0x61C
0xB0000008	...whatever...	...whatever...	jr $ra

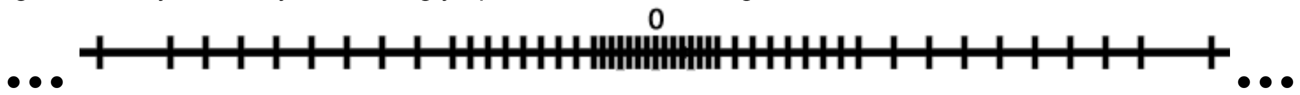
Can't use "j 0xB0000004" in 0xAFFFFFF8, since can't jump across 256MB line (0xA..->0xB..)

c) You can replace the first instruction with a *new* instruction and save 2 clock cycles on a single-cycle non-delayed branch MIPS machine. What is it (in MIPS)? *Careful!* beq $0,$0,2

Name: ________**Answers**____________________ Login: `cs61c-`____

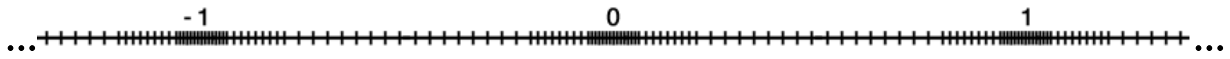
M2) Michael Moore should do his next film on *this* encoding!

Normal floating-point numbers are "bunched up" around zero so that they can represent very small numbers with high accuracy, but they increasingly spread out as the magnitude of the number increases:



Rough sketch of IEEE 754 Floating-point representation number distribution

However, now instead of having accuracy around zero *only*, we would like accuracy around *every* integer. We'll call them "floating point integers," or `flint`s.



Rough sketch of `flint` representation number distribution

We will have *four* fields in our **16-bit** `flint` representation: a mini_`int` part (a signed integer representation), and a mini_`float` part (complete with the usual sign bit, exponent field, and mantissa field):

Integer part	Float sign	Exponent	Mantissa
8 bits (signed representation)	1 bit	4 bits	3 bits
"mini_`int` part"	*"mini_`float` part"*		

The first thing to notice is that mini_`int`s are the same as `char`s in value [-128,127]. That may be handy later.

`Value = value_of(`*mini_`int` part*`) + value_of(`*mini_`float` part*`)`

We decode the raw bit representation into a number using the formula above. The mini_`float` pattern will *not* have ± ∞ nor NaNs, but it *will* have denorms. We wish to adjust the exponent field so `abs(`mini_`float)` < ½. This allows us to "copy & paste" this mini_`float` distribution around every integer (via the mini_`int` *offset*) to get the overall `flint` representation number distribution sketched above. Cool, eh?

A. What should the value of `bias` be? (write your answer in decimal)

$\boxed{17}$

We want the max value to be less than ½ which means
1.111_2 x 2^Y < $.1_2$ (but as close as we can get to $.1_2$)
So how much to shift 1.111_2 to the right to get below 0.1_2?
If Y is -2_{10}, 1.111_2 x 2^{-2} = $.01111_2$, 1/32 less than $.1_2$ (½ - 1/32 = 15/32) so Y=-2
So if 1111_2 – bias = 15_{10} - bias = -2, then bias = 17_{10}. If Y=-2, max(mini_float)=15/32

B. Given the field widths in the diagram and the `bias` value from (A), complete the table. Expressions are OK, but the rightmost column answers **MUST be in base 10!**:

Commentary on number	Bit Encoding	Represented Number (in base 10)
Biggest representable number smaller than 10	`0x` **0A81**	**$10-2^{19}$**
Nothing special, just a nice number	**Flip(15)=>11110000=add1=>11110001** `0x` **F1F8**	**$-0.25 = 1.000_2 x2^{-2}$(S=1,E=max,M=0)** `-15.25`
Most-negative number (closest to -∞)	`0x` **80FF**	**-128 is 0b10000000(=min(char))** **From (A), max(mini_float)=15/32** **$-128 - 2^{-2}(1.111_2)$** **$= -128 - 0.01111_2 =$ <u>-128 15/32</u>** **<u>Encoding into hex yields 80FF</u>**

Name: ________Answers____________________ Login: cs61c-____

M3) Because I MIPS you, baby, and I don't want to C a thing…

What follows is a self-modifying MAL MIPS function. Read it carefully, and answer the questions below.

A) What is the equivalent C code for `mystery`?
Assume for this part that we only call `mystery` once.

```
mystery:
        la     $t0, loop
        addu   $v0, $0, $0

loop:   beq    $a1, $0, done
        lw     $t1, 0($a0)
        addu   $v0, $v0, $t1
        lw     $t1, 4($t0)
        addiu  $t1, $t1, 4
        sw     $t1, 4($t0)
        addiu  $a1, $a1, -1
        j      loop

done:   nop
        nop
        nop
        jr     $ra
```

```
// Precondition: 0 < a1 < 2^15
[unsigned] int      [unsigned] int *
____________ mystery (____________ a0, int a1) {
   // Sums the 1st a1 elements in a0; 2 solutions(A, B)
   int sum = 0, i = 0;

   while(a1--)          // A
      sum += a0[i++];   // A (also) "*a0++"

   for(i=0;i<a1;i++)    // B (also) could count down...
      sum += a0[i]      // B (also) "*(a0+i)"

   return sum
} Note: In order to make the function be equivalent to
  the MIPS code, i must be static or global.
```

B) When we call `mystery` the second time with the *same arguments we used the first time*, do we get the same answer? If not, what do we get? (assume it's not an error)

No, we get the sum of the next a1 elements of a0[] (from a1 through 2*a1-1)

C) Replace as few `nop`s as possible by the `done:` label so that `mystery` behaves as it did in part A *every* time it's called, not just the first time.

```
done:   andi $t1, $t1, 0xFFFF0000   ### We need to reset the imm offset to zero
        sw $t1, 4($t0)              ### And put it back where it was supposed to be
        ____________________
        ____________________
        jr $ra
```

There are other ways to do the same as andi to zero-out $t1's lower halfword immediate: i.e., 0xDEADBEEF ==> 0xDEAD0000

```
a) lui $t1, 0x8C89                             # The original instruction value (clever!)
b) lui $t2, 0xFFFF       (then) and $t1, $t1, $t2   # The same as andi, except in TAL
c) and $t2, $t1, 0xFFFF  (then) xor $t1, $t1, $t2   # 0x0000BEEF xor 0xDEADBEEF = 0xDEAD0000
d) srl $t1, $t1, 16      (then) sll $t1, $t1, 16    # Push halfword off the right edge, and go back
```

Name: __________Answers_______________________ Login: cs61c-____

Post-Midterm Questions

F1) Code bugs are terrified of *RA-AI-AID*!!! (Poof!)

1) If you put 5 drives with a mean time to failure (MTTF) of 10 years in a RAID 0 array, what will be the MTTF of the array?

2 years

2) In an interrupt service routine, should the ready bit of a device be checked before accessing it?

No

3) A CPU running some program was found to have a CPI is 2.5 and, on average, 5 ns of CPU time were used *per instruction*. What is the clock speed in Gigahertz?

½ GHz

```
CPUTime(s) = InstructionCount [inst/prog] * CPI [cycles/inst] * ClockTime [s/cycle]
  ⇨  ClockFreq[cycles/s] = 1 / ClockTime[s/cycle] = CPI * (InstructionCount / CPUTime)
  ⇨  ClockFreq
```

4) What are the two largest challenges that prevent parallel programs from achieving perfect speedup equal to the number of processors?

a) **Communication overhead**

b) **The serial portion of the code**

5) What does Professor Patterson (and team) want to put into the hands of every systems researcher very soon?

Working manycore hardware (via the RAMP project)

6) Rather than endlessly trying to create bug-free programs and somehow find users & operators that never make mistakes (or even trying to eliminate the human element altogether), where does Professor Patterson believe we should focus our efforts?

Recovery & Repair (providing graceful "undo")

7) What does the receiver do with a packet if its checksum indicates it was corrupted in transit?

Deletes it

Name: ________**Answers**__________________ Login: cs61c-____

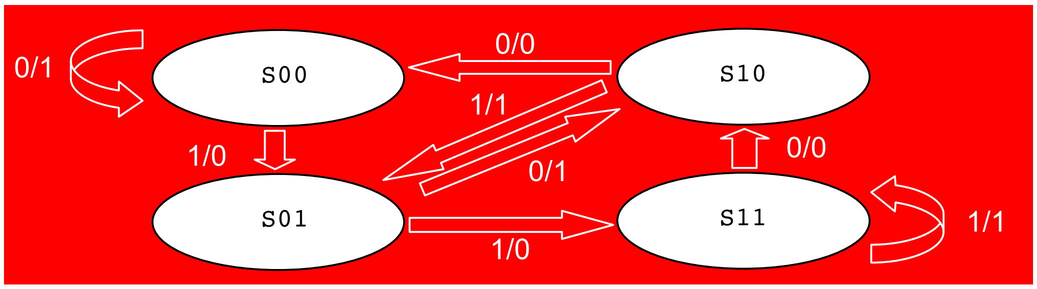
F2) Tune in to 101 on your FSM dial...

We are designing a *palindrome*-finder circuit with a 1-bit input `I(t)` and a 1-bit output `O(t)`, that will produce, at time `t`, whether the sequence `{I(t-2), I(t-1), I(t)}` is the same backwards and forwards (e.g, 101). We'll assume `I(t)` has been `1` for all negative time (i.e., before the finder circuit starts). As an example,

the input:	`I: 1 1 0 0 1 1 1 0 0 0 1 0 1 1 0 0`
will produce the output:	`O: 1 1 0 0 0 0 1 0 0 1 0 1 1 0 0 0`

P1	P0	I	O	N1	N0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	1	1	0
0	1	1	0	1	1
1	0	0	0	0	0
1	0	1	1	0	1
1	1	0	0	1	0
1	1	1	1	1	1

a) Complete the FSM diagram below. Our states have been labeled `Sxy` indicating that the previous 2 bits, `{I(t-2), I(t-1)}` would be `{x, y}`. Fill in the truth table on the right. The previous state is encoded in (`P1`,`P0`), the next state is encoded in (`N1`,`N0`), and the output is encoded as `O`. Make sure to indicate the value of the *output* on your state transitions AND to indicate the starting state with an "incoming arrow".



b) Provide a *fully reduced* (i.e., fewest gates to implement…you can use any *n*-input gates) Boolean expression for the Output `O` as a function of `P1`, `P0` and `I`. If there is a name for the circuit, write it in the box above. E.g., "The always-1", "3-input NAND", etc. A 2-input XOR has the symbol of "⊕".

Name: **XNOR**

$O = \overline{P1 \oplus I}$

c) How many different answers could I have put in the box for "b" above? Said another way, how many different circuits can a 3-LUT imitate?

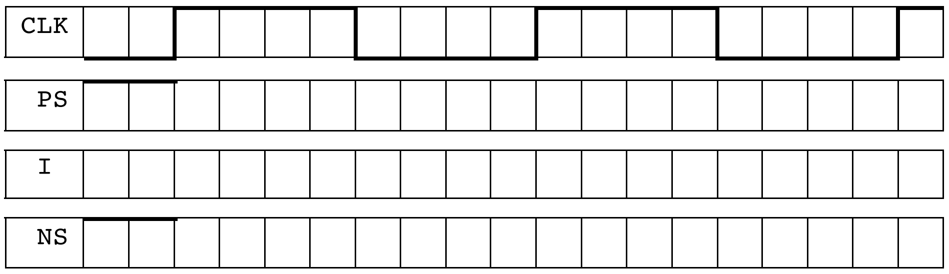
Think of O col as an 8-bit #
How many bit patterns?
256

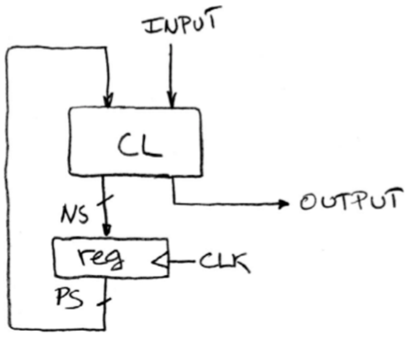
d) We're always concerned about testing. What is the shortest length of an `I(t)` stream that can guarantee you've tested this particular circuit exhaustively?

Here, we're trying to find a path from S11 (initial) through ALL transitions...
8 (10001011)

e) Finally, we wish to build our circuit as we normally do for SDS systems (shown below). Given the four standard spec times from the chip manufacturer (τ_{setup}, τ_{hold}, $\tau_{clk\text{-}to\text{-}q}$, and τ_{CL}), what is the smallest clock period τ we can drive our system with? (Write your answer as an expression involving the spec variables.) Feel free to draw timing diagrams if you wish.

$\tau_{clk\text{-}to\text{-}q} + \tau_{CL} + \tau_{setup} < \tau$,
so to run at max speed,
$\tau = \tau_{clk\text{-}to\text{-}q} + \tau_{CL} + \tau_{setup}$





Name: _________Answers_______________________ Login: cs61c-____

F3) On a final exam, a little cache can boost your performance...

No, we can't be bribed, but we *can* improve our memory access time...let's take a look!

a) For the purposes of this question, assume our MIPS processor has two levels of data caches with the capabilities shown in the table on the right. Also assume:
   - It takes **100 cycles** to go to memory
   - 1GiB of physical address space

   Fill in the table on the right.

	L1	***L2***
Cache Data Size	32 KiB	512 KiB
Block Size	8 B	32 B
Associativity	4-way	Direct-mapped
Hit Time	1 cycle	33 cycles
Miss Rate	10%	2%
Write Policy	Write-through	Write-through
Replacement Policy	LRU	LRU
Tag	17	11
Index	10	14
Offset	3	5
AMAT (in clock cycles)	AMAT L1 = 1 + 0.10 * AMAT L2 = 4.5	AMAT L2 = 33 + 0.02 * 100 = 35

b) In software, you decide to live on the wild side so you turn OFF your L2 cache and clear your L1 cache. Here are other features:
   - 4 GiB of virtual address space
   - 2 KiB page size
   - 8-entry TLB, LRU replacement
   - 32 MiB `ARRAY_SIZE`
   - `char A[]` starts at a block & page boundary (i.e., block, page aligned)
   - The following code is run on the system with no other users and process switching turned off.

```
main() {
    int i,j;
    char *A = (char *) malloc (ARRAY_SIZE * sizeof(char));          // block,page aligned
    for (i = 0 ; i < (ARRAY_SIZE/STRETCH) ; i++) {                  // # of STRETCHes
        for (j = 0 ; j < STRETCH    ; j++)  sum  += A[i*STRETCH + j]; // go up to  STRETCH
        for (j = STRETCH-1 ; j >=0 ; j--)  prod *= A[i*STRETCH + j]; // down from STRETCH
    }
}
```

i. What is number of bits used for the VPN (assume byte addressing)? **21**

ii. What is number of bits used for the PPN (assume byte addressing)? **19**

iii. As we double our `STRETCH` from 1 to 2 to 4 (...etc), we notice the number of cache misses doesn't change! What is the largest value of `STRETCH` *before* cache misses changes? (Use IEC terms, like 64 TiB, 128 GiB, etc.) **32KiB**

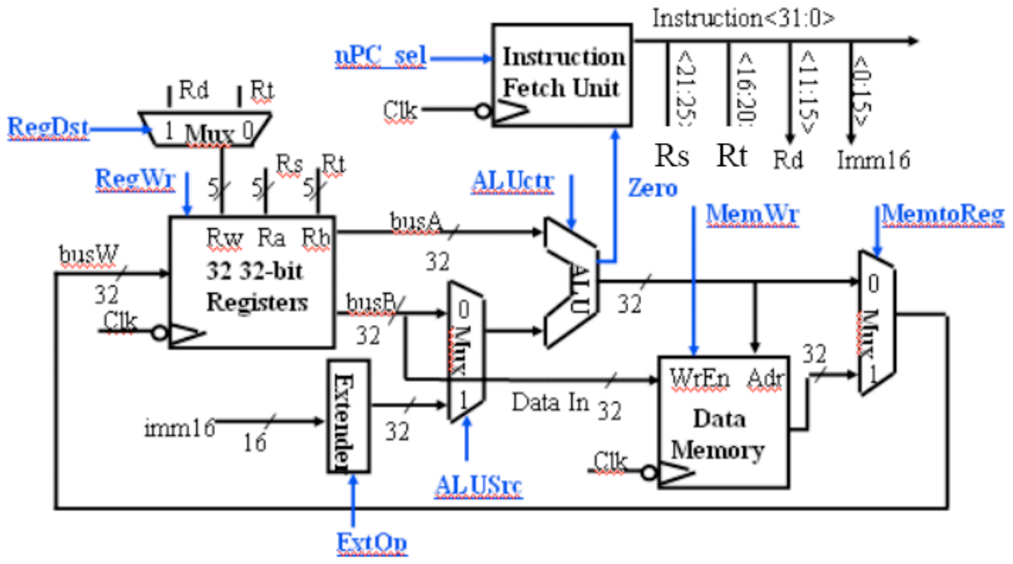
iv. If we double the `STRETCH` from (iii), what is ratio of cache *hits* to *misses*? **29:3**

v. As we double our `STRETCH` from 1 to 2 to 4 (...etc), we notice the number of *TLB misses* doesn't change! What is the largest value of `STRETCH` *before* TLB misses changes? (Use IEC terms, like 64 TiB, 128 GiB, etc.) **16KiB**

vi. For any value of `STRETCH` what is the *fewest number of page faults* we could ever generate? (Use IEC terms, like 64 TiFaults, 128 GiFaults, etc.) **16KiFaults**

Name: __________Answers________________________ Login: cs61c-____

F4) Don't just sit and wait for another datapath you by!



On the right is the **single-cycle** MIPS datapath presented during lecture. ***Ignore pipelining for the question.*** Your job is to modify the diagram to accommodate a new MIPS instruction.

Your modification may use simple adders, shifters, mux chips, wires, and new control signals. If necessary, you may replace original labels.

We want to add a new MIPS instruction (we'll call it `addpr` for "add to pointed reg") that is almost identical to `addi` but with a twist. Instead of storing into the `rt` register the sum of the constant and the value of the register specified by <u>`rs`</u>, it stores into the `rt` register the sum of the constant and the value of the register specified by <u>*the lowest 5 bits in memory at the address specified by the pointer stored in the `rt` register*</u>. Said another way, first get the pointer stored in the `rt` register. Follow that pointer to get its value from memory. Take the lowest 5 bits of that value, treat is as a register number, and find out what is stored in that register. Add that to the immediate, and store it in the `rt` register.

a) Make up the syntax for the I-type MAL MIPS instruction that does it (show an example if the pointer lives in `$v0`, and the constant is 5). On the right, show the register transfer language (RTL) description of `addpr`.

Syntax	***RTL***
`addpr $v0, 5`	`R[rt] = R[MEM[R[rt]](4:0)] + SignExtImm;` `PC = PC + 4;`

b) Change as *little as possible* in the datapath above and list all changes below. You may not need all boxes.

(i)	Add a mux whose output is tied to "Data Memory Adr" and whose input is either the ALU or busB R[rt], driven by a control line called "MemAdr" whose value is either ALU or busB
(ii)	Add a mux whose output is tied to "Ra" and whose input is either Rs or the lowest 5 bits of "Data Memory Data Out", driven by a control line called "RaSrc" whose value is either Rs or Mem
(iii)	

c) We now want to set all the control lines appropriately. List what each signal should be, an intuitive name or {0, 1, x – don't care}. Include any new control signals you added.

RegDst	RegWr	nPC_sel	ExtOp	ALUSrc	ALUctr	MemWr	MemtoReg	MemAdr	RaSrc	
Rt(0)	1	+4	Sign	Imm(1)	Add	0	ALU(0)	busB	Mem	

d) In the context of a single-cycle CPU, `lw` used to be the "critical path" instruction that really determined our fastest clock period, since it utilized the most components of our datapath. Using the terms below to create an expression that will determine *how much slower our clock period will be if we also consider* the `addpr` instruction: `PCRegClkToQ, InstMemAccess, ControlLogicDelay, RegFileAccess, ALUdelay, DataMemAccess, RegSetup, RegHold, MemSetup, MemHold.`

`RegFileAccess` (it has to do that twice. The rest of the terms are the same.)

Name: _________**Answers**______________________ Login: `cs61c-`____

F5) Whose pipeline is it anyway? 500 points if you can guess...

You are the owner of a company that manufactures computers on an assembly line 24 hrs/day. You have 5 stages in your assembly line, and each hour (adjustible if need be) the partially-finished computer moves to the next stage. With 3 shifts of 5 workers working 8 hrs/day, you've been in business for weeks. The 5 stages:

- **PARTS:** Lay out all the parts to be used in the later stages (takes 1 full hour of operator time)
- **HW:** Plug in all the boards and wire it up (takes 1 full hour of operator time)
- **SW:** Install the software (takes 1 full hour of operator time clicking "Ok", "Yes, install everything", etc)
- **TEST:** Test that the machine works (the test always takes 1 full hour, but setting it up only takes a moment of operator time, the rest of the hour the operator sits idle waiting for the test to complete). In the rare case of failure, the test software makes a loud annoying sound and the machine is placed in the parts recycling bin; most of the time the test is successful and the computer moves to the next stage.
- **BOX:** Put the machine in a box, affix a shipping label & load it on a truck (takes 1 full hour of operator time)

a) What is the latency of your assembly line? 5 hours

b) What is the throughput of your assembly line? 1 computer/hour

c) How much faster are you than your competitor who manufactures the same computer but only hires 1 worker per each 8 hr shift? (use Nx, not M% notation) 5x

d) All was going fine until yesterday. Congress passes a law requiring you run a more rigorous *5-hour test* on every machine! This requirement stands to throw a major wrench in your well-crafted manufacturing process. Without hiring anyone else, and with minimal changes to what you already have in place, what do you do to be maximally productive? Draw a diagram to explain yourself.

It might seem that you have to adjust your "clock period" to 5 hours (reducing productivity by a factor of 5), but not so! Add a 5-hour sub-pipeline in the TEST area and you don't have to change your clock! Machines would enter every hour, and take 5 hours to leave. Once you filled this sub-pipeline, you would see (from the point of view of the main pipeline in the abstract sense) that the testing area had the same throughput – a machine would enter and leave the testing area every hour, just like before. [Aside: This is similar to how the lower intestine holds your food to absorb nutrients and water and actually might hold several "meals" in there at once. Your throughput (how often you "go") is the same as if your lower intestine were quite short.]

e) Ok, now back to MIPS. You buy a pimped-up single-user, single-process (aside from the OS), single-CPU, single-core MIPS machine. Miraculously, all 5 pipeline stages *always* complete in exactly 1ns (you spared no expense buying the most cutting-edge parts, especially memory – no cache of any sort was needed since the memory was so fast). A thief breaks into your dorm room and swaps your awesome 1ns memory with cheap 5ns memory. Both memories have the same standard, simple interface (in terms of input and outputs) you saw when we built the MIPS datapath. Your system clock (initially set to 1ns) is adjustable. Does the technique you used in (d) work here? Why or why not?

The sub-pipeline technique would NOT work, because a single memory can't get partially started on 5 different requests. Requests are atomic, so the system has to wait until the previous one finishes before it can start on the next one.

f) How many cycles would it take to execute the first loop iteration for the code on the left on the 5-stage MIPS machines on the right? Note: memory and registers CAN be written and read in the same cycle, branch compares occur in the 2nd stage, we stall on ALL hazards, & there is no out-of-order execution.

Code	*Non-delayed branch* *No forwarding*	*Delayed branch* *Forwarding*
`Loop: lw   $t0, 42($s1)`	`IDEMW        13`	`IDEMW         10`
`      addu $t1, $t0, $s2`	` I  DEMW`	` I DEMW`
`      addu $v0, $t1, $t1`	`    I  DEMW`	`   IDEMW`
`      beq  $t0, $0, Loop`	`       IDEMW`	`    IDEMW`
`      sll  $0 , $0, 0`	`        NNNNN`	`     IDEMW`
	`1234567890123`	`1234567890`