--- 6.004 QUIZ III STUDY GUIDE ---

Contents:
0. Quiz Structure
  1. models of computation questions
  2. stack detective questions
  3. beta hardware questions

Disclaimer: I haven't yet seen quiz 3. This is only what I'd be studying if I was in your shoes. I did look over 6 previous quiz 3s to find the sorts of questions that are asked **over and over again**, so I'd highly recommend using this guide as a place to start your study prep. I tried to make it as condensed as possible.

QUIZ STRUCTURE

You will definitely have a stack detective type question, and a beta hardware question. I imagine the structure will look like this:

1. Models of Computation T/F Questions
2. Extended Stack Detective / Machine Instruction Question
3. Extended Beta Hardware Question.

Any machine instruction questions will likely be similar to the lab5 online questions.

Other Excellent Quiz Review Material:

Online Questions for lab 5: You might be given a code fragment similar to those presented in these online questions, asking you the same sorts of "Value left in R1?" and "Value found in memory location with address 0?" types of questions.

Stacks and Procedures: Questions 1 and 3
[Prof. Terman himself said he loves doing variants of these questions on quizzes.]

Building the Beta: questions 3 and 6
[q3 is a standard "find the bug" type beta question
q6 is a standard "new instruction" type beta question]

Less important material:
"Programmable Machines" tutorial problems
"Models of Computation" tutorial problems -- assuming everything in section 1 of the review packet makes sense to you!
Notes about Finite State Machines (FSMs) and Turing Machines (TMs):

* A TM is strictly more powerful than an FSM: anything an FSM can do, so can a Turing machine. This is because an TM _is_ an FSM, plus more! (An FSM plus an infinite storage tape, namely.)

* In what way is it more powerful? A TM can do things that require unbounded storage, aka unbounded counting, unbounded bookkeeping, etc. For example, a well-formed parenthesis checker requires keeping track of how many more left parens than right parens have been encountered so far. Over all inputs, this number can be arbitrarily high! So an FSM can't do the job, but a TM can because it has infinite storage space.

So TMs get rid of the 'F' in FSMs.

More Review: Church's thesis
"Every discrete function computable by ANY realizable machine is computable by some Turing machine"

This is unproven, and yet widely accepted because nobody has yet built a counterexample. (a physical machine that could, for example, solve the halting problem.)

Problem:
Determine whether each function can be implemented using an FSM, using a universal Turing Machine, or not at all. Circle exactly one answer for each function (answer FSM if the function can be implemented via either a Turing Machine or an FSM.)

1. An unspecified integer function $F(x)$ which is known to be computable on Dan's laptop (with its Pentium and 2 GB of RAM and 80 GB of disk), even when disconnected from external I/O.
   FSM TM uncomputable

2. Given a finite sequence of 1's and 0's representing a binary integer $N$, entered in sequence from high-to-low order, determine whether $N$ is divisible by 3.
   FSM TM uncomputable

3. Given an integer $Y$ between 2000 and 2100, output the number of games of the World Series in the year $Y$.
   FSM TM uncomputable

4. A device that takes as inputs three integers $(I,J,K)$, and which reliably determines if the $I$th TM operating on the $J$th tape will halt within $K$ steps.
   FSM TM uncomputable

5. A device that takes as inputs two integers $(I,J)$ and outputs 1 if the $I$th TM running on the $J$th tape halts; 0 otherwise.
   FSM TM uncomputable
Problem:
Indicate if each statement is:
A) Proved true? 
B) Proved false? or 
C) implied by Church's thesis?

6. The halting function is uncomputable.
Proved ... Implied ... False

7. Every integer function computable by a Turing Machine is computable by some Lambda expression.
Proved ... Implied ... False

8. Intel's Quadraseptium, released in 2142, will compute only those integer functions computable by a Turing machine.
Proved ... Implied ... False

SECTION 2: STACK DETECTIVE QUESTIONS
--- Contract for Procedures [copies from lecture notes] ---
the CALLER will:
* push args onto stack, in reverse order
* branch to callee, putting return address into LP
* remove args from stack on return
the CALLEE will:
* perform promised computation, leaving result in R0
* branch to return address
* leave stacked data intact; including stacked args
* leave registers (except R0) unchanged

--- typical procedure caller / callee code template ---
calling sequence
PUSH(arg_n)          | push args, last arg first
... 
PUSH(arg_1)
BR(f,LP)             | call f
DEALLOCATE(n)       | clean up
... 
callee entry sequence
f: UPDATE(LP)        | save LP and BP,
PUSH(BP)             | just in case we make another procedure call.
MOVE(SP, BP)         | set BP=this frame's base
ALLOCATE(nlocals)    | allocate locals
(push any registers that will be used)
...callee body goes here ...

callee exit sequence
(pop those registers that were pushed)
MOVE(result, R0)    | set R0 to contain the return value
MOVE(BP, SP)        | 
POP(BP)             | restore the caller's base pointer,
POP(LP)             | and linkage pointer.
JMP(LP,R31)         | return to the caller
1. What is the missing C expression corresponding to the `???` in the above C program?

2. What value should appear in place of `???` in the instruction labeled "zz:" in the assembly program?

3. Suppose the instruction bearing the tag "xx:" were eliminated from the assembly language program. Which of the following would result? There's only one right answer:

   A: The program no longer works properly
   B: The program continues to work, but uses more stack space
   C: The program continues to work, with no disadvantages
4. Same question, but for the instruction bearing the tag "yy:")
   A: The program no longer works properly
   B: The program continues to work, but uses more stack space
   C: The program continues to work, with no disadvantages

5: The procedure f is called from an external procedure, and its execution is interrupted just prior to the execution of the instruction tagged "e:". The PC is currently at address 0x54.

The contents of a region of main memory are shown below:

<table>
<thead>
<tr>
<th>ADDRESS (HEX)</th>
<th>CONTENTS (HEX)</th>
</tr>
</thead>
<tbody>
<tr>
<td>B4..............</td>
<td>4</td>
</tr>
<tr>
<td>B8..............</td>
<td>14</td>
</tr>
<tr>
<td>BC..............</td>
<td>0</td>
</tr>
<tr>
<td>C0..............</td>
<td>1</td>
</tr>
<tr>
<td>C4..............</td>
<td>0</td>
</tr>
<tr>
<td>C8..............</td>
<td>3</td>
</tr>
<tr>
<td>CC..............</td>
<td>78</td>
</tr>
<tr>
<td>D0..............</td>
<td>???</td>
</tr>
<tr>
<td>D4..............</td>
<td>1</td>
</tr>
<tr>
<td>D8..............</td>
<td>4</td>
</tr>
<tr>
<td>DC..............</td>
<td>2</td>
</tr>
<tr>
<td>E0..............</td>
<td>78</td>
</tr>
<tr>
<td>E4..............</td>
<td>D4</td>
</tr>
<tr>
<td>E8..............</td>
<td>FFFF</td>
</tr>
<tr>
<td>EC..............</td>
<td>3</td>
</tr>
<tr>
<td>F0..............</td>
<td>0</td>
</tr>
<tr>
<td>F4..............</td>
<td>64</td>
</tr>
<tr>
<td>F8..............</td>
<td>E8</td>
</tr>
<tr>
<td>BP-&gt; FC...........</td>
<td>FFFF</td>
</tr>
<tr>
<td>100..............</td>
<td>2</td>
</tr>
</tbody>
</table>

HINT: this is a "stack detective" type question. before you even LOOK at the questions asked of you, it will help you to label each line in the stack. always do this first.

5a: what is the argument to the current (most recent) call to f?
5b: what value is in SP?
5c: what was the argument to the original (external) call to f?
5d: what is the missing value marked '????' for the contents of location 0xD0?
SECTION 3: BETA QUESTIONS

An EXCELLENT and easy drill to get yourself up to speed on the beta hardware is to try to fill in the basic signals on your own. I’ve made it easy for you by giving you an empty table you can print out. With practice (no memorization necessary!) you should be able to do this in under 5 minutes.

Control logic:

<table>
<thead>
<tr>
<th>OP</th>
<th>OPC</th>
<th>LD</th>
<th>ST</th>
<th>JMP</th>
<th>BEQ</th>
<th>BNE</th>
<th>LDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALUFN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WERF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BSEL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WDSEL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RA2SEL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PCSEL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ASEL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WASEL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
There are two types of Beta questions you will likely be asked: "find the bug" type questions, and "new instruction" type questions.

Find The Bug type

Cost-cutting steps undertaken by the heralded One Beta per Child program have led to five recurring hardware defects, marked in the following diagram:

Each defect causes a signal (or group of signals) to be stuck at zero, independently of the logic driving it. The affected signals are (A) the PC+4 input to the WDSEL MUX; (B) the RA2SEL control signal; (C) the data output from memory; (D) the "1" input to the BSEL MUX; and (E) the output from the adder indicated in the diagram.

Miraculously, you have found that there is never more than one defect in a single Beta.

Your task is to review the following customer complaints, and dignose which if any of the above defects is the likely cause. Enter the appropriate letter of a plausible defect, or NONE if the complaint cannot be explained by one of the above stuck-at-zero faults. Each answer is worth one point.
Find the Bug, continued:

The LD instruction always seems to read zero data
Your Diagnosis: _____

LD, ST and SUBC don’t work
Your Diagnosis: _____

BR(tag,LP) branches to the wrong address instead of tag
Your Diagnosis: _____

Interrupt handlers returns to the wrong address
Your Diagnosis: _____

ST seems to write the wrong data
Your Diagnosis: _____
“New Instruction” type beta question

Marketing has asked for the following instructions to be added to an Extended Beta instruction set, for implementation on the Beta you’ve implemented in the lab:

**LDX( Rx, Ry, Rz) | Load double-indexed**
- EA <- Reg[Ry] + Reg[Rx]
- Reg[Rz] <- Mem[EA]
- PC <- PC + 4

**STX( Ra, Rb, Rc ) | Store indexed**
- EA <- Reg[Ra] + Reg[Rb]
- Mem[EA] <- Reg[Rc]
- PC <- PC + 4

**MVZC(Ra, literal, Rc) | Move constant if zero**
- If Reg[Ra] == 0 then Reg[Rc] <- SXT(literal)
- PC <- PC + 4

**JIPCC(Rx, constant, Rz) | Jump and Increment by PC+Const**
- Reg[Rz] <- Reg[Rz] + 4*SEXT(literal) + PC + 4
- PC <- Reg[Rx]

**SWAPR(Rx, Ry) | Swap register contents**
- TMP <- Reg[Rx]
- Reg[Rx] <- Reg[Ry]
- Reg[Ry] <- TMP
- PC <- PC + 4

**MSWP(Ra, C, Rc) | Swap register contents with memory**
- EA <- Reg[Ra] + SXT(C)
- tmp <- Mem[EA]
- Mem[EA] <- Reg[Rc]
- Reg[Rc] <- tmp
- PC <- PC + 4

Which instructions can be implemented on the existing Beta? For the table below, either fill in the appropriate values for the control signals, or put a line through the whole row if the instruction cannot be implemented using the existing beta datapath. Use “-” to indicate a don’t care value for a control signal.

<table>
<thead>
<tr>
<th>Instr</th>
<th>ALUFN</th>
<th>WERF</th>
<th>BSEL</th>
<th>WDSEL</th>
<th>WR</th>
<th>RA2SEL</th>
<th>PCSEL</th>
<th>ASE</th>
<th>WASEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MVZC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JIPCC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SWAPR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSWP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>