CMSC 311: Computer Organization

Fall 2002

Exam 2 Solutions

CMSC 311 Computer Organization
Fall 2002
RTfP

Quiz 3 (Sequential Logic): "State Your Assumptions" (40 points)

3.1 (2 pts) How much information does a single flip flop store?

Answer: 1 bit

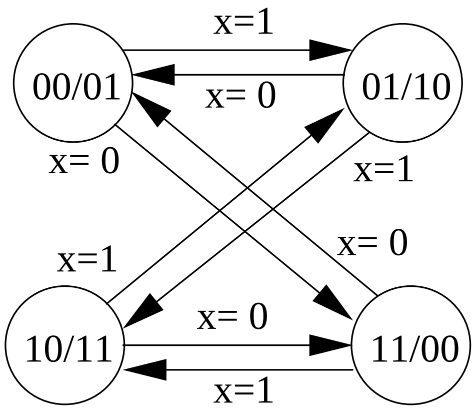
3.2 (2 pts) What does sequential logic provide that combinational logic does not?

Answer: It has "state", i.e., it has memory and can compute output values based on inputs and state.

3.3 (3 pts) Write the excitation table for a JK flip flop.

Answer: Done in class.

3.4 (10 pts) Here's a finite state machine. The states are written (in the circles) as q/z. That is, the state value is written first, followed by the output. x is the inputs.



Fill in the following chart. Some columns do not have labels. You should fill in their names. Use a T flip flop to implement q_1 and a JK flip flop to implement q_0.

Answer: Answer not currently available.

x	q_1	q_0	q_1^+	q_0^+	z_1	z_0			
0	0	0							
0	0	1							
0	1	0							
0	1	1							
1	0	0							
1	0	1							
1	1	0							
1	1	1							

3.5 (7 pts) Draw the circuit from problem 3.2. Use a ROM to implement a circuit. Label CLEARLY. Use flip flops as described in 3.4.

Answer: Answer not currently available.

3.6 (10 pts) Draw a 3-bit parallel-load register with the following operations. Clearly label the inputs, outputs, and parts. You may use logic gates in addition to flip flops.

c_1c_0	Operation
00	Hold
01	Parallel load ($q_3q_2q_1q_0 = x_3x_2x_1x_0$)
10	Negate (two's complement)
11	Flip bits

Answer: Done in class

3.6 (2 pts) Explain the difference between a Mealy and a Moore machine.

Answer: Moore machine has output that depends only on state variables. Mealy machine has output that depends on state and input variables.

(Both machines have basically the same "power", i.e., they are nearly equivalent).

3.7 (4 pts) The behavior of a flip flop can be described by a finite state machine. Draw the finite state machine corresponding to a T flip flop. In particular, think about what the inputs, state and outputs are for a flip flop.

Answer: Done in class

Quiz 5 (Cache): "Cache Us, Clay!" (30 points)

4.1 (3 pts) Given s slots and b bytes in a data block and m slots per set, write a formula determining how many bits are used for tag, slot/set and offset for a fully-associative cache. Clearly label which formula applies to which part (i.e., tag has X number of bits where X is a formula).

Answer: Offset has $\lg b$ bits. Tag has $32 - \lg b$ bits. No bits are used for slot/set.

Note: It's assumed that s and b are powers of 2 (otherwise, getting bits from the address would be much more difficult).

4.2 (3 pts) Given s slots and b bytes in a data block and m slots per set, write a formula determining how many bits are used for tag, slot/set and offset for a direct-mapped cache.

Answer: Offset has $\lg b$ bits. Slot has $\lg s$ bits. Tag has $32 - \lg b - \lg s$ bits.

4.3 (3 pts) Given s slots and b bytes in a data block and m slots per set, write a formula determining how many bits are used for tag, slot/set and offset for a set-associative cache.

Answer: Offset has $\lg b$ bits. Set has $\lg(s/m)$ bits. Tag has $32 - \lg b - \lg(s/m)$ bits. The only tricky part is s/m.

m is not the number of sets. The number of sets is the total number of slots divided by slots per set (use dimensional analysis to convince yourself this is true).

4.4 (4 pts) Explain why cache is used.

Answer: Cache is fast, small memory that stores a frequently used subset of physical memory (RAM). It's used to place frequently used data in faster memory.

4.5 (4 pts) Explain what spatial locality is. Explain what temporal locality is.

Answer: Spatial locality is the idea that if address X is accessed, then addresses $X \pm \Delta$ where Δ is small, are likely to be accessed (e.g., accessing nearby instructions or nearby data in an array).

Temporal locality is the idea that if address X is accessed now, it's likely to be accessed in the near future.

4.6 (4 pts) Suppose you have 16 bytes in a data block, and 32 slots. You are given address A_{31-0}. Indicate which bits are used for the tag, slot, and offset (for a direct-mapped cache).

Answer: Offset uses A_{4-0}. Data uses A_{8-5}. Tag uses A_{31-9}.

4.7 (3 pts) Words (32 bit quantities) are accessed at word aligned addresses (i.e., divisible by 4). Suppose this were not the case, i.e., you could access words at any address. Explain any affect this might have on accessing data stored in a cache.

Answer: This question produced many answers of the form "you wouldn't know where data started and ended".

However, I had a different answer in mind. When getting data from RAM to a cache block, you access a block of data. As long as the block contains at least 4 bytes, then any word aligned data always fits in the block being fetched (this block is also called a *cache line*). If the word is not aligned, it's possible that the word spans two different cache lines.

For example, suppose each cache line stores 16 bytes. Thus, the last 4 bits are used for an offset. If you wanted to access a word whose address ends in 1111, then the word will have one byte in one cache line, and 3 bytes in a different cache line, thus making it difficult to access.

4.8 (6 pts) Suppose you have a 10 bit address A_{9-0}. You are using a fully-associative scheme. Assume the slot has 256 bytes. Implement a circuit that determines whether a slot has the data block corresponding to the address.

The circuit should outputs a 1 if a specific slot contains the data and 0 if it does not. You can use any logic gates, etc. needed. MAKE SURE YOU LABEL CLEARLY or credit will not be given.

Answer: There are two tag bits. Call it T_1 and T_0. The following is a logical expression which you can make a circuit out of:

$$(T_1 \text{ XNOR } A_9) \times (T_0 \text{ XNOR } A_0) \times V$$

XNOR is used to determine when two bits have the same value. V is the valid bit. Many people forgot about the valid bit.

Quiz 6 (MIPS assembly): "Meesh Inspires Popular Support" (30 points)

6.1 (4 pts) What are the "special registers" in MIPS? (Give the index of each register which is special, e.g. $r3). Explain what each of these special registers do.

Answer: $r0 is hardwired 0. $r29 is the stack pointer (by convention). $r31 is the jump-and-link register.

6.2 (5 pts) Translate the following C code to MIPS assembly language. Assume that $r1 is i, $r2 is j, $r3 is k. Comment each line of code.

```
k = 9 - (i + j)
```

Answer: The MIPS code

```
add  $t0, $r1, $r2   # t0 = i + j
addi $t1, $r0, 9     # t1 = 0 + 9 = 9
sub  $r3, $t1, $t0   # r3 = t1 - t0 = 9 - (i + j)
```

6.3 (5 pts) You wish to call a subroutine named FOO. This subroutine will use register \$r1. Before calling FOO, save \$r1 on the stack, then call FOO, then once FOO returns, copy the value from \$r1 to \$r2, and restore the value from the stack back to \$r1. (This is an example of caller-saved subroutine). Note: there is no push or pop instruction in MIPS.

Answer: The MIPS code

```
addi $sp, $sp, -4    # Decrease stack pointer for push operation
sw   $r1, 0($sp)     # Store r1 onto stack
jal  FOO             # Call FOO subroutine
addi $r2, $r1, 0     # r2 = r1 + 0
lw   $r1, 0($sp)     # Get old value of r1 from stack
addi $sp, $sp, -4    # Increase stack pointer for pop operation
```

Note: `subi` is not a MIPS instruction. Use `addi` with negative immediate values.

6.4 (5 pts) Translate the following into MIPS assembly. You must short-circuit logical operations. Assume that \$r1 is i, \$r2 is j, \$r3 is k.

```
if ( i == j || i == 7 )
   j++ ;
else
   k -= 7 ;
i = j - k ;
```

Answer: The MIPS code

```
           beq  $r1, $r2, IF_PART    # if ( i == j ) go to if body
           addi $t0, $r0, 7          # t0 = 7
           bne  $r1, $t0, ELSE_PART  # if ( i != 7 ) go to else body
IF_PART:   addi $r2, $r2, 1          # j++
           j END                     # jump over else
ELSE_PART: addi $r3, $r3, -7         # k -= 7
           sub  $r1, $r2, $r3        # i = j - k
```

6.5 (5 pts) Write MIPS assembly to determine if bit i of register \$r1 is 1 (the other bits may have any other value). If so, set \$r2 to 1, otherwise, set it to 0. \$r3 contains the index i. Comment each line of code.

Answer: The MIPS code

```
      addi $t0, $r0, 1    # t0 = 1
      sllv $t0, $t0, $r3  # t0 << r3
      and  $t1, $r1, $t0  # Check to see if bit i is set
      beq  $t1, $r0, ELSE
      addi $r2, $r0, 1    # r2 = 1
      j END               # jump over else
ELSE: addi $r2, $r0, 0    # r2 = 0
END:
```

`sll` only shifts left a constant value. You need `sllv` to shift a variable amount (or write a loop to shift one bit at a time).

Because `sllv` wasn't covered in class, I accepted `sll` with three registers, even though it is incorrect to do so.

6.6 (3 pts) You can't load a 32 bit immediate value using one instruction. Write instructions to load a 32 bit immediate value. Assume the value you wish to load is $0x1111FFFF$. Comment each line of code.

Answer: The MIPS code

```
lui $r1, 0x1111         # Load upper 16 bits with 0x1111
ori $r1, $r1, 0xFFFF    # Load lower 16 bits with 0xffff
```

Note: `addi` causes sign extension to occur, and produces an incorrect result. However, `addiu` should work since it does not sign-extend (it is an unsigned addition).

6.7 (5 pts) A branch instruction performs a modification of PC + 4 (the next address) if the condition is true. The maximum range of the jump is $(PC + 4) - X$ to $(PC + 4) + Y$ where $(PC + 4) - X$ and $(PC + 4) + Y$ are addresses in memory.

Determine what X and Y are and EXPLAIN how you got those values. (If you don't explain, you receive no credit).

Answer: X is $2^{15} \times 4$. Y is $(2^{15} - 1) \times 4$.

Branch instructions are I-type, which has 16 bits for immediate values. For branch instructions, the representation is signed two's complement. You multiply by 4, because all instructions are word-aligned, so it makes sense to jump based on instructions (4 bytes each) rather than jumping based on addresses.

Challenge ("Drool Kitty") Section: "Meow!" (40 points)

C1 (5 pts) You are given k states and m bits used for the output. Assuming you have a Moore machine, how many possible different outputs are there? Explain your answer.

Answer: $\min(k, 2^m)$.

The number of output depends on m and k. Since it's a Moore machine, you can have at most k different outputs. However, if m is small, there are 2^m different outputs.

For example, if we have 1024 states, but 2 bits of output, then there's only 4 different outputs, despite a large number of states.

C2 (10 pts) You are given a single D flip-flop, plus logic gates. Design a JK flip flop from this. Explain how you designed it.

Answer: You would draw a JK flip flop as a finite state machine, and implement using a single D flip flop.

One person used an idea similar to parallel load registers. They fed in the "next" values into a 4-1 MUX and used J and K as control inputs to the MUX.

C3 (10 pts) Write a recursive function in MIPS assembly that computes the sum of an integer array. Assume that the address of the zeroth element of the array is stored in \$r1. Assume that the size of the array is stored in \$r2, and is valid (i.e., greater than or equal to 0). The final sum should be placed in \$r3.

To solve this problem, you should update the stack in the following way. When you make a recursive call, push the address of the array and the size onto the stack (instead of passing them via registers). When the recursive call is complete, the address and size should be popped off, and the value of the sum placed on the stack. The calling function will then pop off the value from the stack and use it in the computations.

You should not modify \$r1 and \$r2. However, you may modify other registers. Comment each line of code.

Answer: This is a mess. I may post this up Sunday.

C4 (5 pts) A ROM has k-bits for control inputs. You can select the contents of an address. Assume you have a ROM with 3-bits of input, and each address stores 3 bits. Implement a ROM, using any parts (recall exam 1, part means logic gate, or something like a half-adder, etc).

Answer: Use three 8-1 MUXes. Each MUX will have one of the three bits. In effect, the MUX represents one column of the ROM.

C5 (10 pts) Suppose, instead of using "bits", you use "trits". Trits are values in base 3, instead of base 2 (0, 1, and 2 instead of 0 and 1). This system not only affects how the data is stored (in base 3), but also affect the addresses (also in base 3).

Explain what property should govern how many slots and how large the data blocks should be so that you can conveniently use a 32-trit address for caching. Assume a direct-mapped scheme.

Then, pick a specific number of slots and number of trytes (bytes in base 3) in a data block and demonstrate how to break up a 32-trit address A_{31-0} up to conveniently access the parts for cache.

Answer: You should make the number of slots and the number of trytes in a data blocks powers of 3.

For example, if you have 27 slots (which is 3^3) and each slot has 9 trytes (which is 2^2), then A_{31-5} is the tag, A_{4-2} is the slot number, and A_{1-0} is the offset.

The addresses store base 3 values.