COMP2300 Supp/DA Final Exam

Digital Logic Fundamentals 1

Q1. During a work assignment, you are asked to design a combinational circuit with a three-bit input, {A, B, C} (A is the most significant bit and C is the least significant bit), and two 1-bit outputs, O and E. The value of each output is determined as follows:

- The output E is 1 if the input number is even. When the input combinations are all 0, the output is 1
- The output O is 1 only when the input 3-bit number is odd

Answer the following questions.

A. Draw the truth table for the combinational circuit.

B. Express the output E as the simplest sum of products representation. Show your work step-by-step. Your work should first show the expanded Boolean equations translated from the truth table.

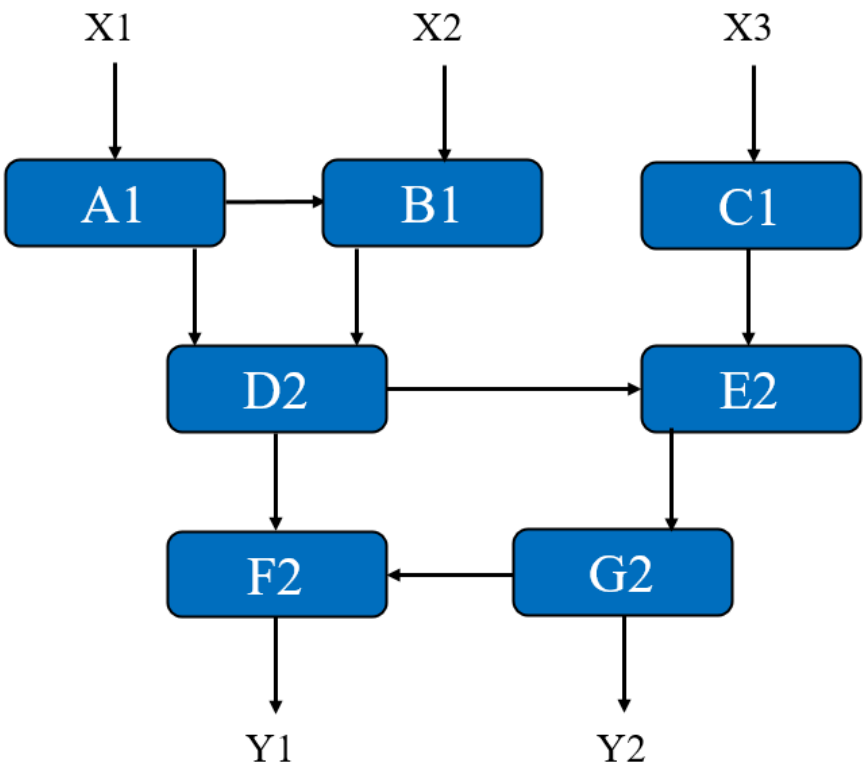
C. Express the output O as the simplest sum of products representation. Show your work step-by-step.

D. Draw a schematic of E and O using basic logic gates.

E. Explain the difference between a multiplexer and a decoder. Which one of the two has a higher logic complexity?

Digital Logic Fundamentals 2

Q2. Consider the high-level schematic of a general-purpose hardware circuit shown that computes two outputs from three inputs. Circle the correct answer. Each question carries 4 points. We will subtract 1 point for each incorrect answer and award 0 points for unanswered questions.



A. Y2 depends on:
    a. X3
    b. X1 and X3
    c. X2 and X3
    d. X1, X2, and X3
B. Y1 has no relationship to X3:
    a. True
    b. False
C. If C1 has a delay of 10 units, and all other components have a delay of 3 units, then the critical path of Y1 consists of:
    a. A1, D2, F2
    b. A1, B1, C1, F2
    c. C1, E2, G2, F2
    d. A1, B1, D2, E2, G2, F2
    e. B1, D2, F2
    f. None of the above
D. If C1 has a delay of 5 units, and all other components have a delay of 3 units, then the critical path of Y1 consists of:
    a. A1, D2, F2
    b. A1, B1, C1, F2
    c. C1, E2, G2, F2
    d. A1, B1, D2, E2, G2, F2
    e. B1, D2, F2

f. None of the above

E. If the input is changed from (X1 =0, X2 = 1, X3 = 0) to (X1 = 1, X2 = 1, X3 = 1), then the following components are triggered to recompute their outputs:
   a. A1, B1, C1, D2, E2, G2, F2
   b. C1, E2, G2
   c. A1, B1, D2, F2
   d. C1, E2, G2, F2
   e. A1, B1, E2, G2, F2
   f. None of the above

Finite State Machines 1

In this question, you will design a finite state machine with the following specification.

- There are two inputs to the state machine, A1 and A2. A1 and A2 are either 1 or 0 at the start of a clock cycle.
- The machine has one output, O1. Initially, O1 is zero.
- O1 is 1 when the XOR of A1 and A2 equals A1 for at least three clock cycles. Otherwise, O1 is 0.
- Assume once O1 transitions to 1, it stays 1 forever.

Design the finite state machine controller by finishing the following tasks.

A. Design the Moore state transition diagram(s) for the above specification. Build the encoded state transition tables. Specify the next state and output equations.

Note: You do not need to simplify the equations or draw the schematic of the resulting circuit.

B. Explain the impact of the sequencing overhead on the clock frequency at which the state machine could be clocked?

Finite State Machines 2

A specialized pipelined processing unit (PPU) for a high-performance computational problem is shown below. The PU processes an input token (set of inputs) using three stages. Your job is to compute the time it takes to process several tokens for different scenarios discussed below.

First, the timing specification for the combinational and sequential elements of the circuit is given below.

- The propagation delay of Stage 1 is equal to 4.7 nanoseconds.
- The propagation delay of Stage 2 is equal to 4.4 nanoseconds.
- The propagation delay of Stage 3 is equal to 4.3 nanoseconds.
- The clock-to-Q propagation delay (t_{pcq}) is equal to 0.4 nanoseconds.
- The setup time (t_{setup}) is equal to 0.1 nanoseconds.
- The hold time (t_{hold}) is zero.

Now, consider the following scenarios. Assume that in the case of multiple PPUs, tokens are evenly partitioned among the PPUs. How long does it take to process the specified tokens?

A. Twelve tokens and one PPU

B. One billion tokens and one PPU

C. One billion tokens and four PPUs

Instruction Set Architecture 1

Part A

Suppose you are using a CPU that implements the base QuAC ISA from the labs, except it has extensions to support the stack and functions. In particular, we've extended register codes to 3 bits and added two new registers, r6 and r8, corresponding to the stack pointer and link register respectively (sp and lr). There are also the four following new instructions with the semantics specified below:

```
------------------

push ra
sp := sp - 1
[sp] := ra

------------------

pop ra
ra := [sp]
sp := sp + 1

------------------
```

```
bl label
lr := pc
pc := label

------------------

bx lr
pc := lr

------------------
```

Implement an iterative Factorial function defined as below **IN QUAC ASSEMBLY**. You do not need to implement the mul function and can assume it already exists.

In order to deal with the fact that mul may clobber registers, invent your own calling convention (which you can assume mul's implementation will follow). Add a comment explaining which registers are caller vs callee saved under your convention. Also ignore numerical overflow; you can assume all results will fit in 16-bit registers.

```
// takes argument n in r1, returns in r1
int factorial(int n) {
  if (n == 0) return 1;
  int acc = 1;
  for (int i = 1; i < n; i++) {
    acc = mul(i, acc);
  }
}

// suppose this function has already been defined and takes arguments
x and y in r1 and r2 respectively, returns in r1
int mul(int x, int y) {
  for (i = 0; i < y; i++) {
    x += x;
  }
  return x;
}
```

Part B

In the last question, we defined multiplication in software using a function since the QuAC ISA does not have a MUL instruction. Explain the advantages and disadvantages of creating an operation using a function in software compared to creating an instruction for the operation in the ISA. Assume a single cycle CPU for simplicity.

Instruction Set Architecture 2

Circle whether each of the following is an aspect of the ISA or the microarchitecture. We will subtract 1 point for each incorrect answer and award 0 points for unanswered questions. Each question carries 2 points.

1. Number of pipeline stages
2. What addressing modes are available
3. Use of an explicit link register for storing return address
4. The power dissipated by processor cores
5. Number of logical/architectural registers
6. Number of physical registers
7. Hardwired versus microprogrammed control
8. Number of bits allowed in memory addresses (size of address space)
9. Physical memory capacity
10. Width of pipeline registers
11. Hazard detection and mitigation
12. Special instructions to access cache lines
13. ALU capability (i.e., number of operations it can perform at once)
14. Range of ALU instructions available
15. Number of issue queue entries in out-of-order processor

Input/Output 1

You are a teacher and have written a program to automatically mark students' work, but are finding it unacceptably slow. The program is 80% calculating marks and 20% collecting those marks into reports. You estimate you can spend 4 hours to make mark calculation 1.1 times as fast, or spend 6 hours to make report creation twice as fast. What is the percentage increase in performance per hour of work for each option, and which would you choose?

Input/Output 2

Say that you want to turn on an LED on your Microbit using the GPIO unit using memory mapped I/O, like how you did in labs. Assume that the hardware the Microbit uses for MMIO is as shown in the below schematic.

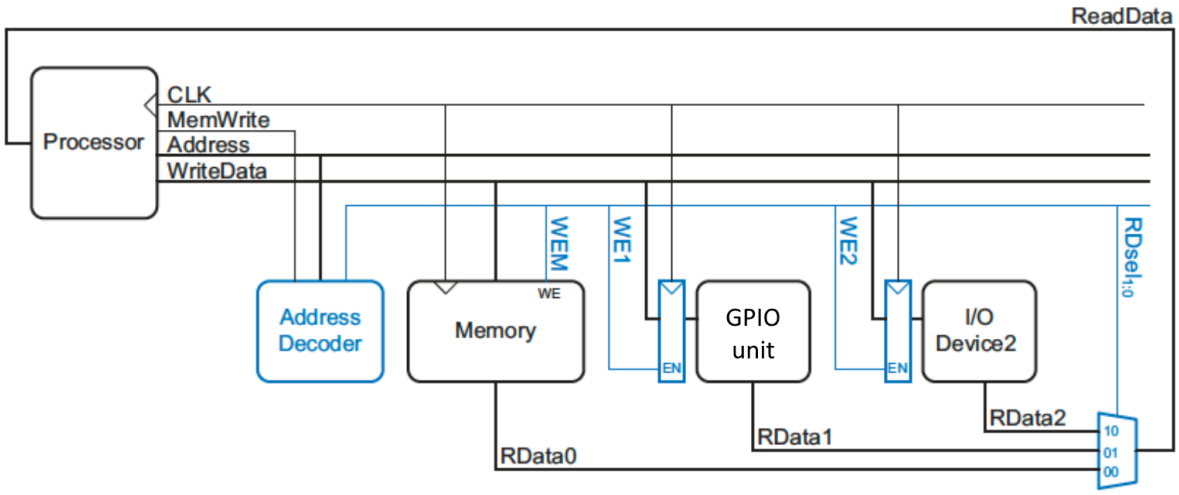


Figure e9.1 **Support hardware for memory-mapped I/O**

Explain what is involved in turning on an LED on the (1) software side and (2) hardware side. This means discussing what instructions you use in software and what memory addresses you might write to, and then how the hardware interprets those instructions and sends the correct signals to the I/O unit.

Microarchitecture 1

Q1. Pick all features from below that impact instruction count.

1. Instruction complexity
2. VLIW (very long instruction word)
3. Complex memory addressing modes
4. Compiler
5. Number of programmer visible registers
6. Transistor quality
7. Pipeline depth
8. Type of stack (downward / upward)
9. Programmer's desk arrangement
10. Levels in the memory hierarchy

Microarchitecture 2

This question requires you to answer questions related to the execution of the following instruction sequence. First, study the instruction sequence below. Then, answer the questions.

Note: You can answer these questions using either text or an illustration.

```
i1:   MOV R0, #0
i2:   MOV R1, #2
i3:   MOV R2, #6
i4:   MOV R3, #19
i5:   MOV R4, #0
     LOOP:
i6:   CMP R1, R2
i7:   BGE EXIT
i8:   ADD R0, #1
i9:   ADD R1, R0
i10:  B LOOP
     EXIT:
I11:  MOV R2, #200
i12:  LDR R1, [R2]
i13:  ADD R3, R1
```

A. How many times is i7 executed if the above code runs on a multi-cycle CPU?

B. Consider the ARM 5-stage pipelined CPU without forwarding and hazard detection in hardware. Your task is to insert NOPs in the assembly code so that it executes correctly on the pipelined CPU. Rewrite the code above by inserting NOPs. What is the IPC if the code with NOPs is executed on this pipelined CPU? (Recall that IPC is acronym for instructions per cycle.)

C. Now consider the ARM 5-stage pipelined CPU with forwarding and hazard detection. In the case of conditional branches, this CPU predicts branches are not taken. You can make the following assumptions.

- Conditional branches resolve and update the PC at the end of EX stage.
- Unconditional branches stall fetch until the PC is updated in the WB stage.
- Memory accesses take one cycle.

For one execution of the loop body (i.e., in some iteration before the loop terminates), identify a pair of instructions where forwarding helps avoid stalls altogether. Also identify a pair of instructions which lead to one or more stall cycles despite hazard detection. What is the IPC if the code executes on the pipelined CPU with forwarding and hazard detection?

D. Assume the CPU uses a more sophisticated branch predictor than always untaken. What is the prediction accuracy of the $Smith_2$ predictor (2-bit saturating counters) for the BGE instruction if the above code executes only once? What if the above code executes four times? Assume the initial state of the Smith predictor is 11 (strongly taken).

Microarchitecture 3

Consider the following ARM assembly code. Answer the following questions.

```
i1: add r1, r0, #10
i2: sub r2, r1, #20
i3: ldr r1, [r0, #10]
i4: add r3, r2, #30
i5: sub r2, r1, #20
i6: str r2, [r0, #10]
```

A. List all the true and false (anti and output) dependencies in the above ARM assembly code. You can list dependencies using the following format: {i1 to i1, anti, by way of R0}.

B. How many independent chains of instructions are there in the code? Write the chains in the following format {i1, i2, i3}.

C. What is the maximum number of instructions that can execute in parallel on a hypothetical machine with no structural hazards (unlimited ALUs, any number of memory accesses in flight at once) and register renaming to deal with false dependencies? Briefly explain your reasoning.

Microarchitecture 4

This question requires you to show the contents of different structures of an out-of-order processor that uses a scoreboard for resolving read-after-write hazards. Use the following assumptions.

(include the same assumptions as in the main exam version of this question, except:)

- The value at memory address 7 is 20
- Contents of register file (and scoreboard/issue queue) are as below:

Scoreboard

	v
R0	1
R1	1
R2	1
R3	1
R4	1
R5	1
R6	1
R7	1

Register file

	value
R0	#8
R1	#7
R2	#6
R3	#5
R4	#4
R5	#3
R6	#2
R7	#1

Issue Queue (IQ)

v	dst tag	rs1 rdy	rs1 tag/value	rs2 rdy	rs2 tag/value
0					
0					
0					

For the four instructions below, answer the following questions.

```
i1: ldr r2, [r1, #0]
i2: sub r3, r2, #6
i3: add r4, r1, #5
i4: add r6, r3, r3
```

Part A

Draw and attach the cycle-by-cycle pipeline diagram, as shown in the lecture slides.

Then, provide the contents of the issue queue at the end of cycle # 8 and cycle # 12. You can pick any free slot in the issue queue for any instruction. Use the following format to answer this question using text.

Cycle # X: {entry 1: v=1, dst tag = R0, rs1 rdy = 0, rs1 tag/val = R0, rs2 rdy = 1, rs2 tag/val = 12}

Part B

Provide the contents of the scoreboard at the end of cycle #5 and cycle # 9. You can show the scoreboard contents in each cycle by writing a string of 8 binary digits, e.g., 00001111 for each cycle.

Part C

Provide the contents of the Register File at the end of cycle # 12. Use the format below.

{R0 = 10, R1 = 10, R2 = 13, and so on for the remaining registers}

Part D

What is broadcasted on the common data bus (CDB) at the end of cycle # 9?

Bonus

The CDC Scoreboard from lectures introduces dynamic scheduling and out-of-order execution which is a significant step up from the in-order architectures explored up to that point, and unlocks new potential for executing instructions in parallel. However, it still suffers from two significant issues:

- It cannot support branch speculation or any other kind of speculation, because it cannot recover from mis-speculation.
- It cannot execute instructions with false (WAR and WAW) dependencies in parallel even though the instructions are not actually dependent on each other in these cases.

Explain, with examples, why the CDC scoreboard has the above issues (e.g. discuss what would happen if it speculates a branch and later finds it was mis-speculated). Then go on to briefly explain how the ARF+ROB architecture introduced in lectures resolves these two problems.