CSE320 Final Exam Practice Questions

Single-Cycle Datapath/ Multi-Cycle Datapath Adding instructions

Modify the datapath and control signals to perform the new instructions in the corresponding datapath. Use the minimal amount of additional hardware and clock cycles/control states.

Remember:

- When adding new instructions, don't break the operation of the standard ones.
- Avoid adding ALUs, adders, Reg Files, or memories to the datapath
- You can add MUXes, logic gates, etc. but try to do minimally. (these cost in terms of area, cycle time, etc)

a. Load Word Register (uses R instruction format)
   lwr Rt, Rd (Rs) #Reg[Rt] = Mem[Reg[Rd]+Reg[Rs]]
b. Add 3 operands (new instruction format: opcode(6), rs(5), rt(5), rd(5), rx(5), (6 bits not used))
   add3 Rd, Rs, Rt, Rx #Reg[Rd] = Reg[Rs] + Reg[Rt] + Reg[Rx]
c. Add to Memory (new instruction format: opcode(6), rs(5), rt(5), rd(5), offset(11))
   addm Rd, Rt, Offset(Rs) #Reg[Rd] = Reg[Rt] + Mem[sign extended offset + Reg[Rs]]
d. Branch on less than or Equal (uses I instruction format)
   blez Rs, label # if Reg[Rs] < 0, PC = PC+4 + (sign-extended offset << 2)
e. Branch Equal to Memory (new instruction format: opcode(6), rs(5), rt(5), rd(5), offset(11))
   beqm Rd, Rt, Offset(Rs) # if Reg[Rt] = Mem[Offset+Reg[Rs]], PC = PC + 4 + Reg[Rd]
f. Branch Equal to 0 to Immediate (uses R instruction format)
   beqzi (Rs), Label #if Mem[Reg[Rs]] = 0, then PC = PC + (sign-extended offset)
   (NOTE: This is not PC+4, and not shifted by 2)
g. Store Word and Increment
   swinc Rt, offset(Rs) #Mem[Reg[Rs] + sign extended offset]= Reg[Rt], Reg[Rs] = Reg[Rs] + 4
h. Store Word and Decrement
   swdec Rt, offset(Rs) #Mem[Reg[Rs] + sign extended offset] = Reg[Rt], Reg[Rs] = Reg[Rs] – 4

What if you were to add (g) and (h) simultaneously to the datapaths?

Datapath Timing

1. Calculate the delay in the modified datapaths when performing instructions above. Assume the following delays:

   - Memory: 200ps
   - Register Files Access (READ/Write): 50ps
   - ALU and adders: 100ps
   - Logic Gates and Multiplexors: 1ps
   - All other times are negligible

2. Calculate the minimal clock cycle time if all of the new instructions were added in the Single and Multicycle cases.

Other Datapath Questions

Given MIPS code, can you determine…..

- Whatis happening at clock cycle X in the Single Cycle Datapath? Or what cycle is operation X happening?
- What is happening at clock cycle X in the Multi Cycle Datapath? Or what cycle is operation X happening?
- How many cycles it will take to execute the code?
- Can you identify the signals (control and values) in the datapath for a given clock cycle?
- And other questions of this nature….

Short Answer Misc Questions

1. What is the primary advantage of fixed-sized opcodes?
   Instruction decode is faster and more efficient. Control does not need to determine the length/ position of the opcode in the instruction.

2. Will a speedup of 20 on 50% of a program result in an overall speedup of at least 2 times? Explain your answer
   The new overall speedup is calculated according to Amdahl's Law. For an overall speedup of 2, the new execution time must be 50% or less of the old execution time.
   No. The new overall execution time = 50% + 50%/20 = 52.5% of the old.

3. What are the 5 components of a modern computer system (Hint: Two of them can be combined and called the processor)
   Datapath + control = processor, memory, input, output
4. What is a stored program computer?
   A computer where the instruction of the program are stored in memory, the CPU is assigned the task of fetching the instruction from memory, decoding them and executing them.

5. True or False:
   - Program execution time increase when the instruction count increase (IC) TRUE
   - In a load/store architecture, the only instructions that access memory are load and store types. TRUE
   - More powerful instructions lead to higher performance since the total number of instructions executed is smaller for a given task with more powerful instructions. FALSE
   - An add operation has 3 operands (2 input and 1 output), therefore add instructions must be 3-address instructions. FALSE
6. In a system executing jobs, when is throughput = 1/latency?
   Throughput of a machine is the number of instructions which are executed per second. Latency is the length of time per execution of an instruction.

   Throughput = 1/latency when a system is executing one task at a time eg. In a single or multi-cycle datapath

Piplelining increases throughput since multiple instructions are executing simultaneously. Therefore the latency of each instruction (on average) is shorter than the length of an instruction.

7. What are the advantages and disadvantages of write-through and write-back cache modifications in shared-memory systems?
   Write-through will slow the system down, taking more time for each write. However, with a write-back cache, there may be data contention since the multiple references could be referencing the data when it is dirty.

Pipelining

1. What are the main benefits and disadvantages of pipelining?
2. Name the type of pipelining hazards. Define how and when they can occur in systems (in general). Define how/when they occur in MIPS. Give a MIPS datapath or code example of each type.
3. Many processors have 5 or 6 stage pipelines. A typical value for the CPI (cycles per instruction) in such processors is in the range of 1.0 to 1.5. Does it mean that the *latency* of execution of most instruction 1 or 2 clock cycles? Why, or why not?
4. Why do conditional branches impact the performance of a pipelined implementation?
5. Briefly describe 2 solutions to reduce the performance impact of conditional branch instructions in a pipelined implementation.
6. Given sequences of instructions determine the forwarding paths and required stalls

Performance

1. The computer spends 82% of the time computing and 18% waiting for the disk. The instruction mix and the average cycles per instruction (CPI) for each type is:

Type	Instruction %	CPI
int	40%	1
FP	30%	5
Other	30%	2

   a. Consider 3 modifications to the computer. Compute the speedup for each.
      i. The processor is replaced with a new one that reduces the total computation time by 35%.
         Speedup = 1 / ((1 - 0.82) + 0.82 * 0.65) = 1.40

      ii. The disk is replaced with a solid state device that reduces the disk waiting time by 85%.
         Speedup = 1 / ((1 - 0.18) + 0.18 * 0.15) = 1.18

      iii. The processor is replaced with a new one that has improved floating point performance. The average floating point CPI is reduced to 3; all other aspects are unchanged.

Average CPI (old) = 0.40 * 1 + 0.30 * 5 + 0.30 * 2 = 2.5
Average CPI (enhanced) = 0.40 * 1 + 0.30 * 3 + 0.30 * 2 = 1.9
Speedup (computation) = 2.5 / 1.9 = 1.315
Speedup = 1 / ((1 - 0.82) + 0.82 / 1.315) = 1.244

b. Which modification gave the best speedup?

Modification (i) provides the best speedup.

c. For the two modifications in part (i) that did not result in the best speedup, is it possible for them to achieve the speedup achieved by the modification in part (ii)? Show your work and explain your answer.

i. An infineitely fast dask: Speedup = 1 / (1 -0.18) = 1.22 which is still slower than i

ii. If FP only 1 clock cycle:

Average CPI (enhanced) = 0.40 * 1 + 0.30 * 1 + 0.30 * 2 = 1.3
Speedup (computation) = 2.5 / 1.3 = 1.92
Speedup = 1 / ((1 - 0.82) + 0.82 / 1.92) = 1.647

2. You have two RiSC-16 processors X and Z, with the following characteristics. They are both *multi-cycle* processors, in which an instruction executes in a *variable* number of processor cycles. X and Z execute variations on the same instruction set (RiSC) is the following way:

a. Processor X implements the base instruction set, including LUI. Processor X implements multiplication in software, meaning there is not MULT instruction.

b. Processor Z eliminates the **LUI** instruction in favor of a **MULT** instruction, getting **LUI** functionality from **LW**.

c. Processor Z's **MULT** instruction uses the ALU over & over again in a loop, performing shifts and conditional adds, and requires 80 processor cycles per multiply.

d. Executing one MULT instruction on Processor Z eliminates on average 30 instructions that would be executed on Processor X when implemented in a software. However, Processor Z then need additional ALU functionality which increasing the ALU's critical path from 10ns to 12ns.
Also, Assume the following:
- Cache read/write: 10 ns
- Register file read/write: 8 ns
- ALU operation: 10 ns for processor X, 12 ns for processor Z

Assume the following distribution of instruction types (assume that **LUI** requires 3 cycles):

	Processor X	Processor Z
MULT	0%	5%
LUI	5%	0%
LW	20%	25%
SW	10%	10%
R-Type	45%	40%
BEQ	20%	20%

For example, if processor Z executes 5 MULT instructions out of every 100. For each MULT instruction, processor X executes an additional 30 instructions.

a. Compare the execution times of the two processors.

Execution time = TIC = Cycle time * Instruction Count * Average CPI
Exec time(x) = Tx * Ix * Cx
Exec time(z) = Tz * Iz * Cz
Tx= 10ns; Tz=12ns

If Iz=100, Ix= 95 + (5 * 30) = 245
CPI for each instruction type:
MULT=80 cycles, LUI =3, LW=5, SW=4, R-TYPE =4, BEQ=3

Therefore:
Average CPI for X = Cx = (0.05*3)+(0.2*5)+(0.1*4)+(0.45*4)+(0.2*3)=3.95
Average CPI for Z = Cz = (0.05*80)+(0.25*5)+(0.1*4)+(0.4*4)+(0.2*3)=7.85

Comparing execution times:
Exec time(x) = 10 ns/c * 3.95 c/i * 245 i = 9677.5 ns
Exec time(z) = 12 ns/c * 7.85 c/i * 100 i = 9420 ns

Processor Z with the multiply instruction is about 1.03 times faster than processor X for this instruction mix.

b. At what clock speed for processor Z are the two designs equal in performance?

Equating the two excution times and solving for Tz
10ns * 3.95cpi * 245 instructions = Tz * 7.85cpi * 100 instructions
Tz = (10ns * 3.95 * 245)/(7.85* 100)
Tz= 12.33ns

For smaller Tz (faster clock), processor Z has better performance; for larger Tz (slower clock), processor X has better performance.

c. *(more difficult)* Assuming the original ALU latency for processor Z (12 ns), how fast would your softwareemulated multiply have to be (on average) for processor X to be just as fast as processor Z? In other words, how many instructions would processor X execute in place of 1 MUL?

Remember that Ix was defined to be 95 + (# of multiplications) * (cost of each)
We first need to find the instruction count of processor X necessary for equal performance.
10ns * 3.95 * Ix = 12ns * 7.85 * 100 which implies Ix = 238.5
Total number of multiply-emulate instructions is (238.5-95)= 143.5
Therefore number of instructions per multiply = 143.5/5 = ~28 instructions

3. Two important parameters control the performance of a processor: cycle time and cycles per instruction. There is an enduring trade-off between these two parameters in the design process of microprocessors. While some designers prefer to increase the processor frequency at the expense of large CPI, other designers follow a different school of thought in which reducing the CPI comes at the expense of lower processor frequency. Consider the following machines, and compare their performance using the following instruction mix: 25% loads, 13% stores, 47% ALU instructions, and 15% branches/jumps. Assume the unmodified multi-cycle datapath and finite state machine.
   - M1: The multicycle datapath is designed with a 1 GHz clock
   - M2: A machine like M1 except that register updates are done in the same clock cycle as a memory read of ALU operation. Thus in the finite state machine, states 6 and 7 and states 3 and 4 are combined. This machine has an 3.2 GHz clock, since the register update increases the length of the critical path.
   - M3: A machine like M2 except that effective address calculations are done in the same clock cycle as a memory access. Thus states 2, 3, and 4 can be combined, as can 2 and 5, as well as 6 and 7. This machine has a 2.8 GHz clock because of the long cycle created by combining address calculation and memory access.

   Find out which of the machines is fastest. Are there instruction mixes that would make another machine faster, and if so, what are they?

   In the original multi-cycle data path the CPI for each instruction is as follows:
   Loads: 5 cycles
   Stores: 4 cycles
   ALU: 4 cycles
   Branch/Jumps: 3 cycles

   Performance M1:
   Average CPI = .25*5 + .13*4 + .47*4 + .15*3 = 4.1
   Cycle Time = (CPI*#instructions)/ clock rate = 4.1I/1GHz = $4.1I*10^{-9}$ seconds

   Performance M2:
   Loads shorten to 4 cycles
   ALUs shorten to 3 cycles
   Average CPI = .25*4 + .13*4 + .47*3 + .15*3 = 3.38
   Cycle Time = (CPI*#instructions)/ clock rate = 3.38I/3.2GHz = $1.06I*10^{-9}$ seconds

   Performance M3:
   Loads shorten to 3 cycles
   Stores shorten to 3 cycles
   ALUs shorten to 3 cycles
   Average CPI = .25*3 + .13*3 + .47*3 + .15*3 = 3
   Cycle Time = (CPI*#instructions)/ clock rate = 3I/2.8GHz = $1.07I*10^{-9}$ seconds

   M2 is fastest.

M1 can never be faster than M2, even if all the instructions are branch instructions, the CPI will be 3 for all 3 cases, and the clock rate is faster on the other 2 processors.

M3 can be faster than M2, if all instruction loads or all stores then

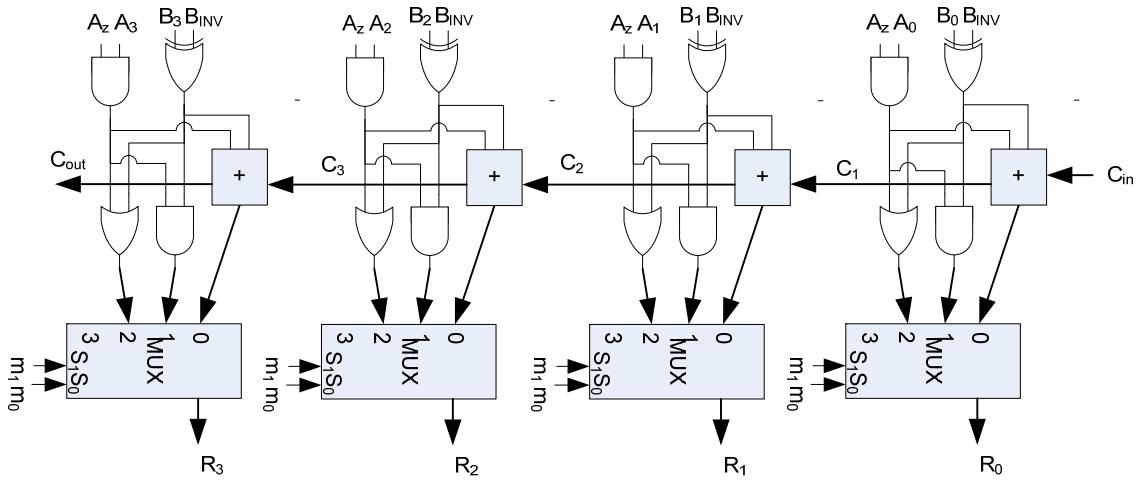
Ex: M2: Average CPI = 1*4 + 0*4 + 0*3 + 0*3 = 4
M3: Average CPI = 1*3 + 0*3 + 0*3 + 0*3 = 3

M2 Cycle Time = (CPI*#instructions)/ clock rate = 4I/3.2GHz = 1.25I*10^{-9} seconds
M3 Cycle Time = (CPI*#instructions)/ clock rate = 3I/3.2GHz = 1.07I*10^{-9} seconds

Review Adder and ALU Creation and Building larger ALUs from units

Consider the 4-bit ALU below which can perform the following 5 operations: add, sub, AND, OR and negate B.

Inputs are A={A_3,A_2,A_1,A_0}, B={B_3,B_2,B_1,B_0}, and C_{in}. Outputs are result R={R_3,R_2,R_1,R_0}and C_{out}. Numbers are in 2's complement form. Fill in the table below, for each operation, what the values of the control signals should be. Indicate don't cares where appropriate.



Operation	m_1	M_0	C_{in}	B_{INV}	A_z
Add	0	0	0	0	1
Sub	0	0	1	1	1
OR	1	0	X	0	1
AND	0	1	X	0	1
Negate B	0	0	0	0	1

Digital Logic

1. Using Boolean algebra, prove the following:

a. bd' + c'd = ((b'd') + (cd))'

bd' + c'd = (b'd')'(cd)' De Morgans
bd' + c'd = (b+d)(c'+d') De Morgans
bd' + c'd = bc'+ c'd+ bd' + dd' Distributive
bd' + c'd = bc'+ c'd+ bd' + 0 Complementary
bd' + c'd = bc'(d+d')+ c'd+ bd' Null
bd' + c'd = bc'd + bc'd'+ c'd+ bd' Commutative
bd' + c'd = c'd (1 + b) + bd'(c' + 1) Distributive
bd' + c'd = bd' + c'd Null

b. abc' + bc'd + a'bd = abc' + a'bd

abc' + bc'd(a+a') + a'bd = abc' + a'bd Null
abc' + abc'd + a'bc'd + a'bd = abc' + a'bd Distributive
abc'(1+d) + a'bd (c'+1) = abc' + a'bd Commutative
abc' + a'bd = abc' + a'bd Null

c. a' + a(a'b + b'c)' = a' + b + c'

a' + a((a'b)' (b'c)') = a' + b + c' De Morgans
a' + a((a+b')(b+c')) = a' + b + c' De Morgans
a' + a(ab+bb'+ac'+bc') = a' + b + c' Distributive
a' + a(ab+0+ac'+bc') = a' + b + c' Complementary
a' + a(ab+ac'+bc') = a' + b + c' Distributive
a' + aab+aac'+abc' = a' + b + c' Idempotence
a' + ab+ac'+abc' = a' + b + c' Idempotence
a' + a(b+c'+bc') = a' + b + c'
a' + b + c'+ bc' = a' + b + c' No Name
a' + b + c'(1+ b) = a' + b + c' Null
a' + b + c' = a' + b + c'

2. Consider the following function: $z(x_3,x_2,x_1,x_0) = x_3'x_2 + x_3x_1x_0 + x_3x_2x_0' + x_3'x_2x_0$

a. How many literals does z contain? **11**
b. Is z, minimal? If not, find the minimal expression using Boolean algebra.
No.

$x_3'x_2 + x_3x_1x_0 + x_3x_2x_0' + x_3'x_2x_0$
$x_3'x_2(1 + x_0) + x_3x_1x_0 + x_3x_2x_0'$
$x_3'x_2(1 + x_0') + x_3x_1x_0 + x_3x_2x_0'$
$x_3'x_2 + x_3'x_2x_0' + x_3x_1x_0 + x_3x_2x_0'$
$x_3'x_2 + x_3x_1x_0 + x_2x_0'$

c. Find the equivalent sum of minterms (SOP) for z (using Σm notation)
$z(x_3,x_2,x_1,x_0) = \Sigma m(4,5,6,7,11,12,14,15)$

d. Find the equivalent product of maxterms (POS) for z (using ΠM notation)

$z(x_3,x_2,x_1,x_0) = \Pi M\ (0,1,2,3,8,9,10,13)$

3. You were recently hired as an engineer in a company that designs alarm systems custom made to meet the customer's specifications. You are asked to design a system that uses the inputs of three sensors A, B, and C. The alarm should go off (activated) when the following criteria are met:

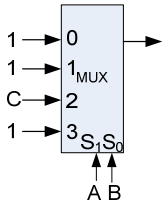
   - When A is off, or
   - When B is on and C is off, or
   - When both A and C are on.

   a. Write the truth table for the function

A	B	C	**Alarm**
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

   b. Write the Boolean expression in Product of Sums (POS) form.
   A' + B + C

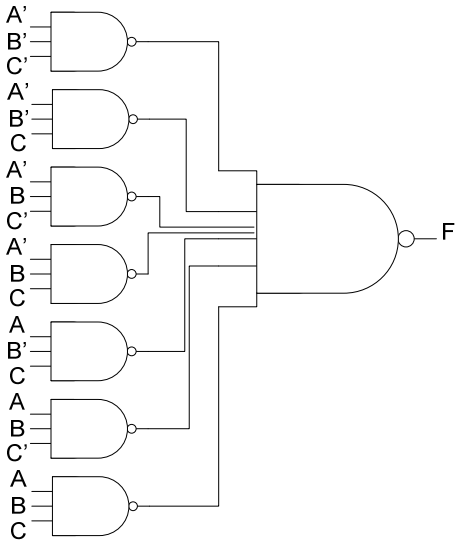
   c. Draw/Implement the function using 2-selector MUX gates.



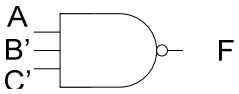
a.

   d. Draw/Implement the function using 2-level NAND-NAND gates.

The straight forward solution using minterms/SOP expression:



A Better Solution using Demorgan's law: **A' + B + C = ((A' + B + C)')' = (AB'C')'**



4. Find the minimal 2-level implementation using NOR-NOR gates, of a system with two 2-bit inputs (A = {a_1, a_0} & B = {b_1, b_0}) which output the following. If A+B is even, then the output is their product. If A+B is odd, then the output is their sum.

a_1	a_0	b_1	b_0	**A+B (decimal)**	**Z (decimal)**	Z_3	Z_2	Z_1	Z_0
0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	1	0	0	0	1
0	0	1	0	2	0	0	0	0	0
0	0	1	1	3	3	0	0	1	1
0	1	0	0	1	0	0	0	0	0
0	1	0	1	2	2	0	0	1	0
0	1	1	0	3	3	0	0	1	1
0	1	1	1	4	3	0	0	1	1
1	0	0	0	2	0	0	0	0	0
1	0	0	1	3	3	0	0	1	1
1	0	1	0	4	4	0	1	0	0
1	0	1	1	5	5	0	1	0	1
1	1	0	0	3	3	0	0	1	1
1	1	0	1	4	3	0	0	1	1
1	1	1	0	5	5	0	1	0	1
1	1	1	1	6	9	1	0	0	1

Z_3 = a1a0b1b0

Z_2 = a1a0'b1 + a1a0b1b0' = a1a0'b1 + a1b1b0'

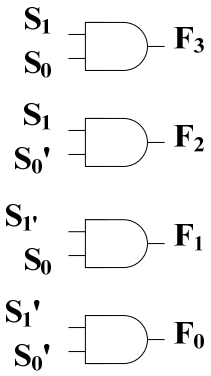
Z_1 = a1'b1b0 + a1'a0b0 + a1'a0b1 +a1a0b1' + a1b1'b0

$Z_0 = (a0 + b0)(a1+a0'+b1$

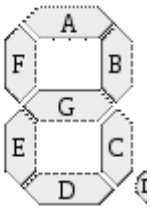
5. Implement the functionality of a 2-input Decoder using minimal AND, OR and NOT gates.

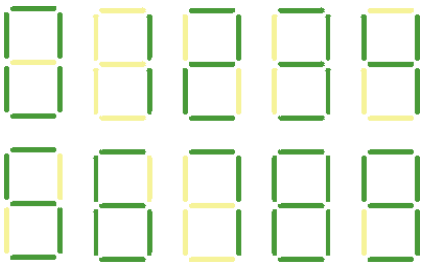
Decoders take a binary number and map this value to an output line. 2-input values, means 4 different values (4 outputs)

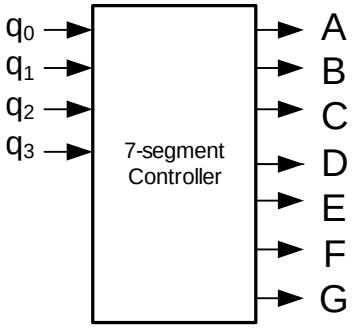
S_1	S_0	F_3	F_2	F_1	F_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0



6. Implement a 7-segment Controller.(truth table, Boolean expressions, gate logic). Practice with any combination of logic units.







(a):

Decimal $q3q_2q_1q_0$	7-segment A B C D E F G
0000	1 1 1 1 1 1 0
0001	0 1 0 0 0 0 0
0010	1 1 0 1 1 0 1
0011	1 1 1 1 0 0 1
0100	0 1 1 0 0 1 1
0101	1 0 1 1 0 1 1
0110	1 0 1 1 1 1 1
0111	1 1 1 0 0 0 0
1000	1 1 1 1 1 1 1
1001	1 1 1 1 0 1 1

(b): $\mathrm{A} = (q_3 + q_2 + q_1 + q_0')(q_3 + q_2' + q_1 + q_0)$

$\mathrm{B} = (q_3 + q_2' + q_1 + q_0')(q_3 + q_2' + q_1' + q_0)$

$\mathrm{C} = (q_3 + q_2 + q_1 + q_0')(q_3 + q_2 + q_1' + q_0)$

$\mathrm{D} = (q_3 + q_2 + q_1 + q_0')(q_3 + q_2' + q_1 + q_0)(q_3 + q_2' + q_1' + q_0')$

$\mathrm{E} = q_3'q_2'q_1'q_0' + q_3'q_2'q_1q_0' + q_3'q_2q_1q_0' + q_3q_2'q_1'q_0'$

$\mathrm{F} = (q_3 + q_2 + q_1 + q_0')(q_3 + q_2 + q_1' + q_0)(q_3 + q_2 + q_1' + q_0')(q_3 + q_2' + q_1' + q_0')$

$$\mathrm{G} = (q_3 + q_2 + q_1 + q_0)(q_3 + q_2 + q_1 + q_0')(q_3 + q_2' + q_1' + q_0')$$

7. Implement the 7-segment using a 4-selector DEMUX and Or gates.