EEE336 Digital Design Spring Semester 2013-14
SOLUTIONS

1. a. i) A floating point number is made from 2 parts:

The mantissa, ***m*** - sets the precision (how exact the number is)

The exponent, ***e*** - sets the dynamic range (how big or small the number is)

The value of a floating point number is $\boldsymbol{m} \times \boldsymbol{2^e}$ **(4)**

ii) Align to the number with the largest exponent (e1) by right shifting the mantissa of the number with the smaller exponent (e2) by (e1 – e2) places. The numbers can then be added as shown.

$m_1 \times 2^{e1} + m_2 \times 2^{e2} = (m_1+(m_2 \text{ shr } e_1\text{-}e_2)) \times 2^{e1}$ if $e_1 > e_2$

$m_1 \times 2^{e1} + m_2 \times 2^{e2} = (m_2+(m_1 \text{ shr } e_2\text{-}e_1)) \times 2^{e2}$ if $e_2 > e_1$

$m_1 \times 2^{e1} + m_2 \times 2^{e2} = (m_1+m_2) \times 2^{e1}$ if $e_1 = e_2$ **(4)**

b. Multiplication will give a 10-bit result including a sign bit.

Sign extending multiplicand gives 0000011101

2s complement of multiplicand gives 1111100011

0000011101

01110(0)

0000000000	first pair of multiplier bits **00** > do nothing
1111000110	second pair of multiplier bits **10** > subtract multiplicand (x2 shift)
0000000000	third pair of multiplier bits **11** > do nothing (x4 shift)
0000000000	fourth pair of multiplier bits **11** > do nothing (x8 shift)
0111010000	fifth pair of multiplier bits **01** > add multiplicand (x16 shift)
0110010110	sum of partial products confirming that 29 x 14 = 406 **(8)**

c. Good for three or more consecutive 1's in the multiplier as Booth encoding replaces three or more adds with one add and one subtract.

For two 1's there is no gain, Booth encoding replaces two adds with one add and one subtract .

For single 1's Booth encoding replaces one add with one add and one subtract

Modified Booth solves this problem by examining multiplier bits in groups of three and performing a single add for the case 010. **(4)**

2. a. At the end of the Fetch-Execute cycle, the control path will examine the interrupt input when deciding whether to fetch the next the next instruction; if the interrupt is high, the processor will jump to an interrupt service routine (ISR). An ISR is just a specialised kind of function call and so the register values, program counter etc. will be saved on the stack. After servicing the external event, the ISR will return, popping the stack as it does so and resume operation from the instruction immediately following the one it was executing when the interrupt was called.

Interrupts are a better way of handling external events since if polling is used, the processor is placed in a continual loop. If the loop contains a large amount of processing as well as testing for the external event, then the response time for handling the interrupt will be long. Alternatively, if the testing loop does little but test for the event (or tests for the external event occur frequently), the processor operation will be inefficient. The basic interrupt operation allows the processor to respond to external events on demand and waste no time in fruitless tests. **(6)**

b. Interrupt latency is the time delay between an interrupt being physically signalled and the processor starting to handle the interrupt. Interrupt latencies can be a critical factor in real-time control applications where proper functioning or indeed safety, may depend on the system being able to respond to an interrupt within some time period. **(3)**

c. It is often difficult to estimate any bound on latency and hence system response to a given interrupt. There is can be a question mark over whether small timing variations in the arrival of an interrupt will lead to unintended system behaviour; testing all combinations of such a scenario is often impractical. **(3)**

d. **i)** The most common instructions were identified as register-to-register moves. Variables and intermediate results can be stored in registers and do not then require repeated loads and stores to/from memory. Procedure calls are also faster as registers can be used for parameter passing. **(3)**

ii) The complexity of the control circuitry is reduced accordingly. **(2)**

iii) The combinational logic delay of the control circuitry. **(1)**

iv) The simpler control circuitry required for RISC operations reduces the critical delay path allowing higher clock frequencies. **(2)**

3. a. The conventional von Neumann architecture has a single memory interface; the same block of memory holds both code and data. This gives flexibility and the simplicity of only requiring one memory interface although as both code and data have to pass through the same interface, this can act as a bottleneck – the so called von Neumann bottleneck.

In the Harvard architecture, code and data memories are separate and each has its own interface. Thus data memory accesses and code fetches can be fully overlapped leading to a greater processing throughput, a factor which is paramount in DSP applications since the rate of processing effectively dictates the bandwidth of the system.

(6)

b. Sign extending each operand to 8 bits gives: 00001110 ÷ 00000101

Initially, we form 00001110 – (00000101 x 2^3) = 00001110 – 00101000, which can be most easily achieved by taking the 2s-complement of 00101000 = 1101100. Therefore, 00001110 – 00101000 = 00001110 + 11011000 = 11100110

(As a check, = -26). Therefore, $q_3 = 0$

The second stage is: 11100110 + (00000101 x 2^2) = 11011101 + 00010100 = 11111010.

(As a check, -6). Therefore, $q_2 = 0$.

The third stage is: 11111010 + (00000101 x 2^1) = 11111010 + 00001010 = 00000100.

(As a check, +4). Therefore, $q_1 = 1$.

The final stage is: 00000100 – (00000101 x 2^0) = 00000100 – 00000101. Taking the 2s-complement of 00000101 gives: 11111011. Thus:

00000100 – 00000101 = 00000100 + 11111011 = 11111111

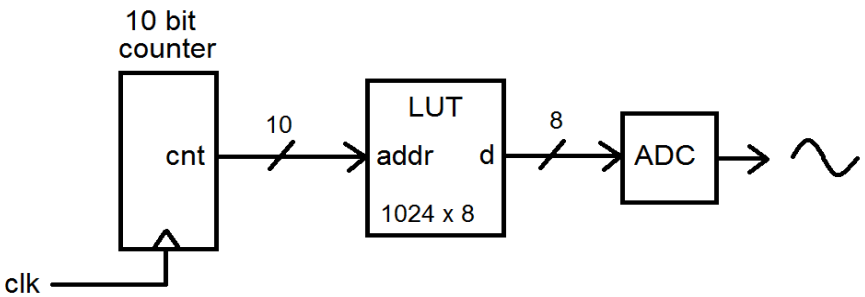
Since this is negative, q1 = 0 and we must restore the dividend to 00000100.

Therefore the answer is $q_3q_2q_1q_0 = 0010$ remainder 0100.

14 ÷ 5 = 2R4

(6)

c.



clk frequency = 1024 x 50 KHz = 51.2 MHz

1k memory requires a 10-bit address bus

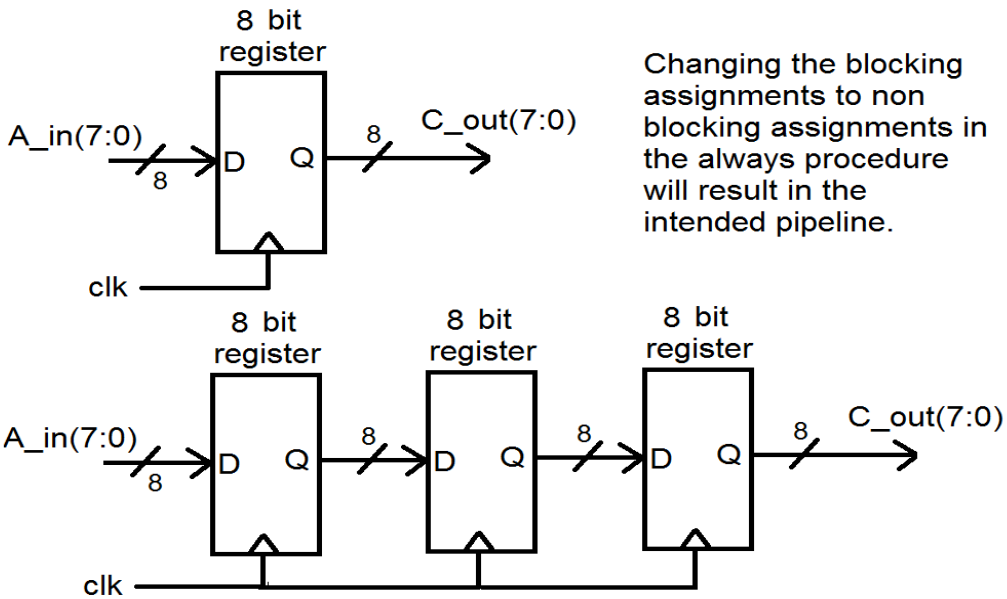
There are two symmetries for a sine wave which can be exploited to reduce the size of the ROM by 4. [sin(π – θ) = sin(θ) and sin(π + θ) = -sin(θ)] **(8)**

4. a. **i)** **wire** – net data type used for connections, does not store any data, must be driven.
reg – variable data type used as a storage element, stores value between assignments, holds its value until a new assignment is made. **(2)**

ii) **initial** procedure – executes once only, never repeats, single-pass behaviour
always procedure – cyclic behaviour, executes whenever there is an event on its sensitivity list **(2)**

iii) **blocking assignment** (=) – execution flow in the procedure is blocked until the assignment is completed, statement must be executed before the execution of the statements that follow
nonblocking assignment (<=) assignment to left hand side is delayed until other evaluations in the current time step are completed, flow not blocked. **(2)**

b.



Pipeline registers can be used to break down combinational circuitry into a number of stages to improve the speed of a circuit. **(4)**

c. States:

S0 – idle state, wait for a start signal to begin

When Start = 1 , reset PP (PP ← 0) , load multiplier (MR) & multiplicand (MD), initialise a counter to the number of bits in the multiplier.

S1 – Check multiplier lsb (MR[0])

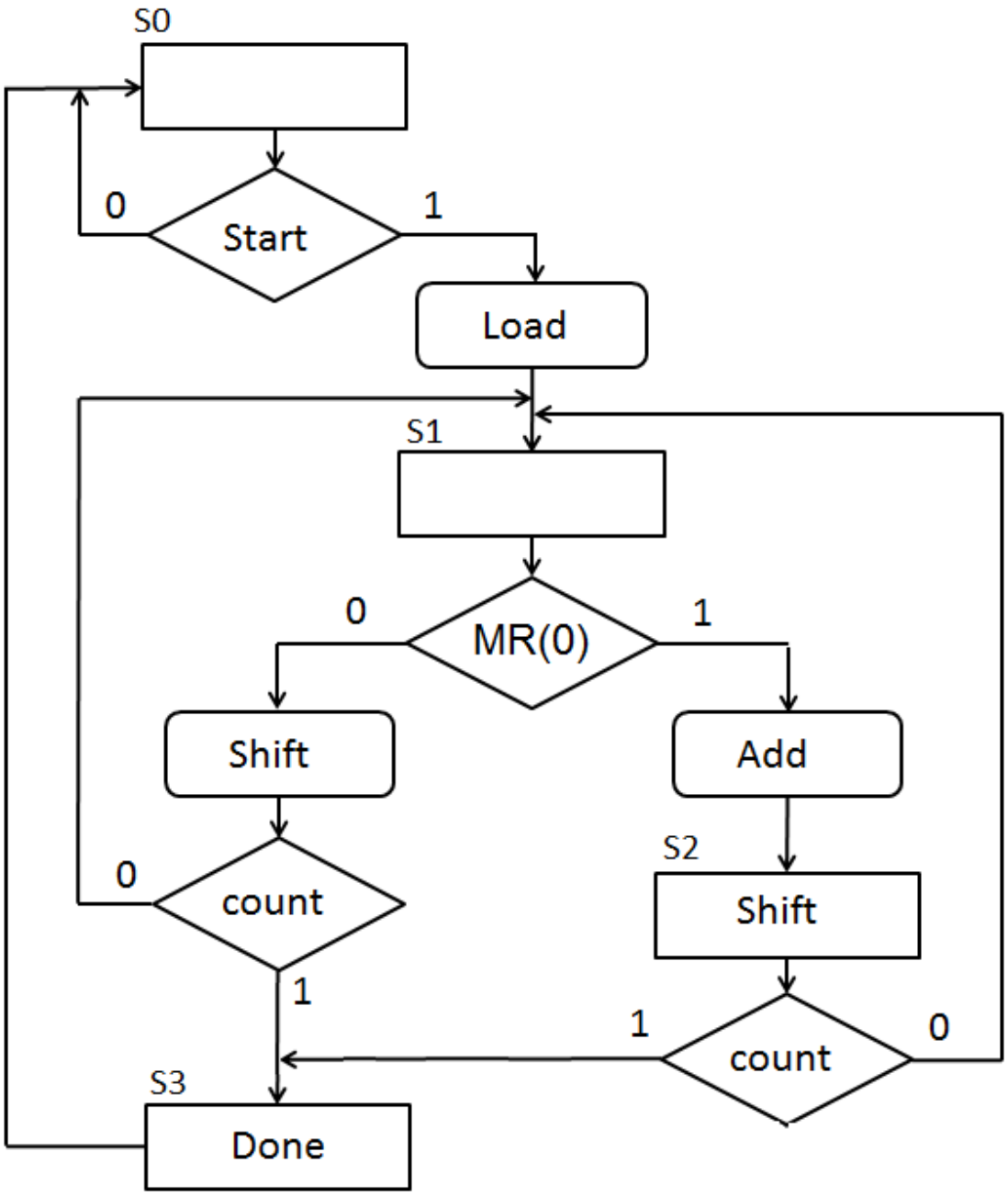
If MR[0] = 0 , perform the right shifts and increment counter, test for final count,

If MR[0] = 1 , add multiplicand to PP go to state2

S2 – perform the right shifts, test for final count

When final count is reached , enter state3

S3 – Output result , set done signal **(8)**


S0
Start
0
1
Load
S1
MR(0)
0
1
Shift
Add
S2
Shift
0
count
1
1
count
0
S3
Done

NJP/ JR