SN2016 Exam: Answers.

Q1

(a) (i) $0032 (ii) $FFF1

(b) This makes it very difficult to detect overflow conditions such as C or V which, in turn, makes it hard to design programs that can modify behaviour depending on such conditions. It is for example difficult to extend arithmetic to word lengths beyond 16 bits.

(c) If $80 and $7F are added the 8-bit result will be $FF. In unsigned code $80 and $7F are 128 and 127 respectively. The sum, $FF = 255, is correct so C =0. In signed code the numbers are -128 and +127 with a sum of -1. This again is correct so V=0. As the result is not 0, Z=0 and as it would be negative if this was a signed code, N =1.

(d) Outline solution…

```
; for (i=0; i<4; i++)
;     {
;      temp = x[i]
;      x[i] = y[i]
;      y[i] = temp
;     }
; Register usage …
      …
;
      ADD       R1,R0,R0      ; i = 0
      LEA       R2,4[R0]      ; R2 = 4
      LEA       R3,1[R0]      ; R3 = 1
Loop  LOAD      R4,X[R1]      ; R4 = x[i]
      LOAD      R5,Y[R1]      ; R5 = y[i]
      STORE     R4,Y[R1]      ; Set y[i] to previous value of x[i]
      STORE     R5,X[R1]      ; And x[i] to previous value of y[i]
      ADD       R1,R1,R3      ; i++
      CMPLT     R15,R1,R2     ; If i< 4 then…
      JUMPT     R15,Loop[R0]  ; Loop
      TRAP      R0,R0,R0

X     DATA      …
      …
Y     DATA      …
      …
```

(e) Outline solution:

```
; for (i=0; i<4; i++)
;     {
;      if(x[i] > y[i]
;          {
;            temp = x[i]
;            x[i] = y[i]
;            y[i] = temp
;          }
;     }
; Register usage …
      …
;
      ADD       R1,R0,R0      ; i = 0
      LEA       R2,4[R0]      ; R2 = 4
      LEA       R3,1[R0]      ; R3 = 1
Loop  LOAD      R4,X[R1]      ; R4 = x[i]
      LOAD      R5,Y[R1]      ; R4= y[i]
      CMPGT     R6,R4,R5      ; If x[i] > y[i] perform swap
      JUMPF     R6,Skip[R0]   ; Otherwise skip (do nothing).
      STORE     R4,Y[R1]      ; Set y[i] to previous value of x[i]
      STORE     R5,X[R1]      ; And x[i] to previous value of y[i]
 Skip ADD       R1,R1,R3      ; i++
      CMPLT     R15,R1,R2     ; If i < 4 then…
      JUMPT     R15,Loop[R0]  ; Loop
      TRAP      R0,R0,R0

X     DATA      …
      …
Y     DATA      …
      …
```


Q2

(a) An exception is an event that causes the CPU to undergo a process break. When an exception occurs, the CPU stops executing the current process and jumps to an exception handler routine. In a system with no OS or a very simple OS, a user process may have to handle any exceptions that occur while it is running . In a machine with a multitasking OS, initial exception handlers are part of the OS and run at OS system levels of privilege . Once the handler has completed, depending on the nature of the exception, control may be returned to the original process or the latter may be aborted . When an OS is handling the exception, once the handler routine has completed, the original process may not be allowed to run immediately even if it is not aborted but may rather be queued by the process scheduler .

(b) (i) Reset is caused by assertion of a reset hardware line input to the CPU. The handler is designed to reinitialised the system and, where appropriate, initiate a new boot of the OS. The process running when the reset occurs is aborted

(ii) Interrupt is caused by assertion of an interrupt hardware line input to the CPU. The line may be asserted by an I/O device which is requesting service or a system timer indicating that synchronous OS intervention is due. The interrupted process is not usually aborted but after the interrupt handler has run, the scheduler may place it in a process queue and return control to a different process.

(iii) A trap is an exception generated by a software instruction. Unconditional trap instructions are used to request OS controlled services for a user program. Such services may include I/O, communication, termination etc.

(iv) A memory fault is an exception usually generated by the MMU via a hardware input to the CPU. The fault may occur in a VM system when a page that is being accessed is not present in memory or in any protected memory system where a process attempts to access an address which is not in an authorised segment. In the former case, a page fetch occurs and the process is reinstated; in the latter, the process is aborted.

(c) (i) When an IRQ interrupt occurs, the program counter will be loaded with whatever 16-bits are currently stored in the interrupt vector (at $FFF8/$FFF9) and execution will then transfer there. If the vector is not initialised, the CPU will jump to whatever address is there which will in general not contain real code. The result will be an almost inevitable crash.

(ii) When an interrupt occurs, the 6811 uses the stack to store not only the return address but the entire programmer's model. This is then restored again when RTI is executed. The 6811 stack grows downwards in memory and must be initialised so that such downward growth is in a region of RAM not used for anything else. If the stack pointer (S register) is not set up, it may be pointing at any area of the address space, so the CPU may be unable to store its state (ROM or unpopulated address) or may overwrite other code or data (RAM).

Q3

(a) The Internet is a store-and-forward inter network where data is held at routers prior to being forwarded along the next hop. Limiting the size of packets makes buffer management at routers easier and aids fairness in the sharing of links. It also makes error detection and correction more reliable.

Each packet carries the address of its destination and routers use this to choose the best output to forward the packet on. The routing tables used by routers are dynamically updated with information gathered from other routers and so the best route to a given destination can change over time. Since every packet is routed independently, two that are in the same conversation can take different routes. As a result, a later packet may on occasion overtake an earlier one.

Another possibility is that packets can sometimes get lost or destroyed in the network. In this case the sender may be asked to retransmit the missing packet which will then arrive late.

(b) If two packets arrive in the wrong order the destination can only do something if it can detect the issue. Such detection is only possible if a suitable transport protocol (most likely TCP) is in use. TCP segments, which are carried one-to-one inside IP packets, are numbered and so out of order arrival is easily detected. When this happens, the TCP software will first notice that there is a problem when a later segment arrives instead of the one expected. The later segment data is placed in a reassembly buffer and the sender asked again for the missing one. If the missing packet now arrives the data in it is inserted into the correct position in the buffer. Once data without gaps has been received it can be passed up to the application.

(c) There are various reasons and any two will do. A packet may get lost due to lack of buffers at a router which then must discard data. Another possibility is a faulty routing decision due to non-convergent forwarding tables, leading to the packet looping and exceeding its TTL. A third is damage to the packet header due to bit errors not spotted by link protocols.

(d) If a packet gets lost the implications depend on the application. In some cases, the missing data must be replaced. This applies in e.g. a file transfer and scenarios like this are best handled with a TCP transport. In other cases, such as streamed audio or video, replacing the data, which is time-critical, does not make sense but such applications are usually tolerant to small losses which appear to a user as clicks or flickers in the streamed media. In cases such as this a UDP transport may be more suitable.