CS1S Computer Systems: Questions and Solutions
May 2018

Duration: 1 hour
Rubric: Answer both questions
This examination is worth a total of 50 marks

1. **(a)** Convert 1010 0010 to a decimal number, assuming binary representation.

[2]

Calculate 128 + 32 + 2 = 162 by adding the powers of 2 corresponding to the positions where there is a 1 bit in the word. [Problem solving.]

(b) Convert 1010 0010 to a decimal number, assuming two's complement representation.

[3]

Since the leftmost bit is 1, this is a negative number. Negate it to get a nonnegative number. To negate, first invert it giving 0101 1101. Then increment it, giving 0101 1110. Now this result is nonnegative so its binary representation is the same as its two's complement value; this is 64 + 16 + 8 + 4 + 2 = 94. Since the negation of the original word is 94, the answer is -94. [Problem solving. 1 mark for identifying it as negative; 2 marks for negation.]

(c) Translate the statement `a := b - c*d` into Sigma16 assembly language, assuming that a, b, c, d are signed integer variables. You do not need to write a complete program, and you don't need to write data statements for the variables. Just translate this one statement.

[5]

```
   load   R1,b[R0]   ; R1 := b
   load   R2,c[R0]   ; R2 := c
   load   R3,d[R0]   ; R3 := d
   mul    R4,R2,R3   ; R4 := c*d
   sub    R4,R1,R4   ; R4 := b – c*d
   store  R4,a[R0]   ; a := b - c*d
```

[Problem solving, requires understanding usage of registers, variables, and basic instructions. 2 marks for loads and store, 1 mark for not loading a, 2 marks for arithmetic.]

(d) Translate the following high level language program fragment into low level language. The variables sum, i and n are signed integers, and x is an array of signed integers containing n elements. You do not need to define the variables or array, just translate the program code. (The low level language contains assignment statements, goto statements, and statements of the form if b then goto label, where b is a Boolean expression.)

```
sum := 0
```

```
i := 0
while i < n && x[i] > 0 do
   sum := sum + x[i]
   i := i + 1
```

[5]

```
  sum := 0
  i := 0
loop:
  if i >= n then goto done
  if x[i] <= 0 then goto done
  sum := sum + x[i]
  i := i + 1
  goto loop
done:
```

[Problem solving, requires understanding while loop translation.]

(e) Translate the program in part (d) into a complete program in Sigma16 assembly language. Use data statements to define the following initial values: n = 4, x[0] = 7, x[1] = 2, x[2] = 0, x[3] = 5. What is the value of sum when the program terminates?

[10]

At termination, sum = 7 + 2 = 9

```
; R1 = sum
; R2 = i = loop index
; R3 = n = array size
; R4 = constant 1
; R5 = x[i]

; Initialise the variables in registers
       add    R1,R0,R0      ; R1 = sum := 0
       add    R2,R0,R0      ; R2 = i := 0
       load   R3,n[R0]      ; R3 := n
       lea    R4,1[R0]      ; R4 := 1

; Traverse the array until stopping condition
loop   cmp    R2,R3         ; compare i with n
       jmpge  done[R0]      ; if i >= n then goto done
       load   R5,x[R2]      ; R5 := x[i]
       cmp    R5,R0         ; compare x[i] with 0
       jmple  done[R0]      ; if x[i] <= 0 then goto done
       add    R1,R1,R5      ; sum := sum + x[i]
       add    R2,R2,R4      ; i := i + 1
       jump   loop[R0]      ; goto loop
done   store  R1,sum[R0]    ; sum := R1
       trap   R0,R0,R0      ; terminate

; Define variables
```

```
n       data   4
sum     data   0
x       data   7
        data   2
        data   0
        data   5
```

[Problem solving, requires understanding of the instruction set, conditionals, indexed addressing and loops. 3 marks for initialization and store, 5 marks for loop, 2 marks for array access]

2. **(a)** Give the truth tables for the following logic gates: and2, or2, xor2.

[3]

a	b	and2 a b	or2 a b	xor2 a b
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

[Bookwork. 1 mark for each gate.]

(b) Consider a circuit that takes two inputs a, b. It produces one output, x, which is 1 if the two inputs have the same value, as shown in the truth table. Implement the circuit using any of the standard logic gates (inv, and2, or2, xor2). You may specify the circuit using any of the following notations (just use one): a schematic diagram, Boolean algebra, or Hydra notation.

a	b	x
0	0	1
0	1	0
1	0	0
1	1	1

[3]

There are many solutions, including

```
x = inv (xor2 a b)
x = or2 (and2 (inv a) (inv b)) (and2 a b)
```

Any correct solution is acceptable. [Problem solving.]

(c) Explain the purpose of the clock in a synchronous circuit. Describe how a suitable clock speed for the circuit is determined.

[4]

The clock generates a sequence of ticks, which define points in time. The clock is connected to every flip flop, and a flip flop updates its state only at the tick, so the effect is to cause all flip flops to update their states simultaneously. Without this, it would be possible for one flip flop to update before another, and its output changes leading to a spurious change of the input to the second flip flop. By synchronizing all flip flops, the entire machine has a single state which is a vector of all the individual flip flop states.

The clock must run slowly enough to allow all combinational signals to settle down to a stable final value before the next tick. This means the cycle must be long enough for the slowest combinational path to settle down; this is the critical path. In addition, some additional time is added to the clock period to allow for random variations in component speeds. [Bookwork. 2 marks for purpose, 2 marks for speed.]

(d) The following program determines the sum of the elements and the number of elements in a linked list, given a pointer p to the head of the list. Each node is a record consisting of two words: the first word "value" is an integer, and the second word "next" is a pointer to the rest of the list. The last node in the list has nil in the next field (nil is represented by 0). Translate the program to Sigma16 assembly language. You don't need to write out the low level language version, and you don't need to define the variables or the linked list.

```
; given p = pointer to list
length := 0
sum := 0
while p /= nil do
   length := length + 1
   sum := sum + (*p).value
   p := (*p).next
done: terminate
```

[10]

```
; R1 = p (given)
; R2 = length
; R3 = sum
; R4 = constant 1
; R5 = temp

     add    R2,R0,R0       ; length := 0
     add    R3,R0,R0       ; sum := 0
loop
     cmp    R1,R0          ; compare p with nil
     jumpeq done[R0]       ; if p = nil then goto done
     add    R2,R2,R4       ; length := length + 1
     load   R5,0[R1]       ; R5 := (*p).value
     add    R3,R3,R5       ; sum := sum + (*p).value
     load   R1,1[R1]       ; p := (*p).next
     jump   loop[R0]       ; goto loop
done
     store  R2,length[R0]  ; length := R2
     store  R3,sum[R0]     ; sum := R3
     trap   R0,R0,R0       ; terminate
```

[Problem solving. 2 marks for dereferencing p, 1 mark each for accessing fields of the node, 4 marks for loop structure, 2 marks for initialization and storing.]

(e) State what the processor does when an interrupt occurs**.** Give two advantages of using interrupts to catch errors (such as overflow or division by zero) rather than using instructions to test explicitly for the error.

[5]

An interrupt is a jump executed by the processor in response to an event, but not as the result of executing a jump instruction. The event may be caused by a program error (such as overflow), or an external signal (for example the timer going off or an I/O device needing service). A process is a running program with evolving state. When an interrupt occurs, the machine jumps from a running process to the interrupt handler. If the interrupt was caused by a program error, and an exception handler has been established, the handler is invoked and it enables the program to decide what to do to recover from the error. An alternative is for the program to check the condition code after each instruction that could cause an error. Advantages of using an interrupt include: it's more robust because it checks every instruction, while the programmer might forget to check the condition code after some operation; it's more efficient because the program doesn't need explicit instructions to check the condition.
[Bookwork. 3 marks for interrupt, 1 mark for each advantage.]