

COMP2300

Introduction to Computer Systems

Study Period: 15 minutes

Time Allowed: 3 hours

Permitted Materials: One A4 page with notes on both sides.

Questions are NOT equally weighted.

The questions are followed by labelled, framed blank spaces into which your answers are to be written. Additional answer panels are provided (at the end of the paper) should you wish to use more space for an answer than is provided in the associated labelled panels. If you use an additional panel, be sure to indicate clearly the question and part to which it is linked.

The marking scheme will put a high value on clarity so, as a general guide, it is better to give fewer terse answers in a clear, succinct manner than to outline a greater number in a sketchy, half-answered fashion.

The Appendix contains information on the PeANUt instruction set.

Name (family name first):

Student Number:

[4 marks]

(b) The IEEE single precision floating point standard is: 1 bit sign, 8 bits exponent with a bias of 127, and the remaining 23 bits are the mantissa. What is “not a number” (nan) and how is it represented?

[1 mark]

The following are for use by the examiners.

Q1 Mark	Q2 Mark	Q3 Mark	Q4 Mark	Q5 Mark	Total Mark
---------	---------	---------	---------	---------	------------

Question 1 (continued)

(c) AND, OR, NAND and NOR are four basic gates that are used in the construction of computer processors. Each gate receives two input signals that are either 0 or 1, and produces a single output signal that is also either 0 or 1. If A and B are defined as the input to these gates and X is defined as the output, complete the following tables

AND		
A	B	X
0	0	
0	1	
1	0	
1	1	

OR		
A	B	X
0	0	
0	1	
1	0	
1	1	

NAND		
A	B	X
0	0	
0	1	
1	0	
1	1	

NOR		
A	B	X
0	0	
0	1	
1	0	
1	1	

[2 marks]

(d) If X, Y and Z are all 16-bit two's complement integers such that:

$$X = 1011\ 1101\ 0001\ 0111$$

$$Y = 1110\ 1001\ 0011\ 0000$$

$$Z = X - Y$$

What is the value of Z. Give your answer as both a two's complement number and as a signed decimal number. Show your working.

[4 marks]

Question 1 (continued)

(e) The original von Neumann machine had five basic devices (or parts). Draw a sketch of the von Neumann architecture, labeling the five basic components and how they interact.

[2 marks]

(f) What is the major difference between the ASCII character set and UCS/UNICODE character set.

[1 mark]

(g) Variable instruction length is one characteristic of a CISC machine. In contrast the original RISC machines had a fixed instruction length. What is the major advantage of having a fixed instruction length?

[1 mark]

Student Number:

Student Number:

Question 2 [20 marks] C Programming Language

(a) The following program manipulates character string data using functions from the standard C library:

```
#include <stdio.h>
#include <string.h>

int main( )
{
    char name1[20]="computer";
    char name2[20]="systems";
    char *name3;
    int n;

    /* printf("First String\n");*/
    n=strlen(name1);
    printf("name1=%s:n=%d\n",name1,n);

    /* printf("Second String\n");*/
    n=strlen(name2);
    printf("name2=%s:n=%d\n",name2,n);

    printf("Final String\n");
    name3=strcat(name1,name2);
    n=strlen(name3);
    printf("name3=%s:number=%d\n",name3,n);

    return 0;
}
```

Relevant details from the strlen/strcat man page are given below:

```
int strlen(const char *s)
The strlen() function returns the number of bytes in s, not
including the terminating null character.

char *strcat(char *s1, const char *s2)
The strcat() function appends a copy of string s2, including
the terminating null character, to the end of string s1. The
function returns a pointer to the null-terminated result.
The initial character of s2 overrides the null character at
the end of s1.
```

Question 2 (continued)

The program compiles and runs without errors. Write down exactly what output is given by this program. Number each line of the output and, if relevant, clearly indicate the presence of each blank space. The library functions produce no output, and there are less than 10 lines of output in total.

[3 marks]

(b) Write C code for a function

```
int mystrlen(char *val);
```

that performs exactly the same operations as the library routine strlen. Your code MUST be less than 10 lines, and should NOT call any other function.

[3 marks]

Question 2 (continued)

(c) Write C code for a function

```
char *mystriecat(char *str1, const char *str2);
```

that performs exactly the same operations as the library routine strcat. Your code MUST be less than 20 lines, and should NOT call any other function.

[6 marks]

Question 2 (continued)

(d) The following program compiles and runs without error.

```
#include <stdio.h>
#include <string.h>
void mystery(char *val, int num);
int main()
{
    char name1[20]="computer";
    int n;

    n=strlen(&name1[2]);
    mystery(&name1[2],n);
    printf("name1=%s\n",name1,n);

    return 0;
}
void mystery(char *val, int num)
{
    int i, j;
    char vali;
    for (i = 1; i < num; i++) {
        vali = val[i];
        j = i;
        while( (vali > val[j-1]) && (j > 0)) {
            val[j] = val[j-1];
            j--;
        }
        val[j] = vali;
    }
}
```

Function mystery should look familiar! Explain BRIEFLY what it is doing.

dl:

[2 marks]

Exactly what is printed out when the program is run?

d2:

[2 marks]

Question 2 (continued)

(e) The following gives two alternative versions of the function `doit`. The function is supposed to accept an integer array as input and returns the sum, maximum and total number of elements in that array. The array is terminated by an integer that has a value of 99 or greater. One version of `doit` is correct, the other is incorrect.

```
int doit1(int *val, int *tot, int *max)    int doit2(int val[], int tot, int max)
{
    int i=0;                            {
    *max = val[i];                       int i=0;
    *tot = 0;                             *max = val[i];
    while (val[i] < 99){                 tot = 0;
        *tot = *tot + val[i];            do {
        *max = (val[i] > *max) ? val[i] : *max;
        i++;                             tot = tot + val[i];
    }                                     if (val[i] > max) max = val[i];
    return i;                             i++;
    }                                     } while (val[i] < 99);
    }                                     return i;
}
```

We wish to call `doit` from the following program.

```
int main( )
{int n,sum,max;
 int val[5];
 val[0]=9; val[1]=1; val[2]=-5;
 val[3]=16; val[4]=100;

 /* call to function doit HERE */

 printf("number of elements < 99 = %d \n",n);
 printf("sum of valid elements = %d \n",sum);
 printf("maximum valid element = %d \n",max);
 return 0;
}
```

Which version of `doit` is correct and how is it called?

e1: [2 marks]

List two fundamentally different problems with the incorrect version of `doit`.

e2: [2 marks]

Question 3 [25 marks] PeANUt

(a) Describe in one sentence each, two disadvantages of machine language code over assembly code.

a1: [2 marks]

a2: [2 marks]

(b) What is the purpose of the Program Status Word (PSW) in PeANUt? What are its two components? For each component, explain in one sentence what it is needed for.

[3 marks]

(c) Assume the value in the accumulator is decimal 42. Which of the following PeANUt assembly instruction do you have to use so that the value in the accumulator after the instruction is decimal 32? Give as answer either A, B, C, D, E or F.

- A: sub 32
- B: sub 10
- C: sub #10
- D: load 10
- E: load #32
- F: sub !10

[1 mark]

Question 3 (continued)

(d) Assume the PeANUit accumulator contains the bit pattern 00001010 11001101, and the instruction sequence `not` or `#255` is executed (255 as a decimal number). Which of the following bit patterns is stored in the accumulator after these instructions? Give as answer either A, B, C, D, E or F.

- A: 00001010 11111111 C: 00001010 00110010 E: 11111111 11001101
- B: 11001101 00001010 D: 11110101 00110010 F: 11110101 11111111

[1 mark]

(e) Write a sequence of PeANUit assembly language instructions that do the same as the following piece of C code. Use `trap #1` to end your program.

```
int i, e, f, s;
s = 1;
e = 8;
f = 1;

i = s;
while (i <= e) {
    f = f*i;
    i = i+1;
}
```

[9 marks]

Question 3 (continued) Student Number:

(f) Assume the accumulator (AC) contains the value 42₁₀, and that memory cell 200₈ contains the value 42₁₀. Describe in detail what happens when the PeANUit machine executes the following instruction which is stored at memory address 100₈. `cmp 2008`

Your answers should describe the events that happen in the three phases *Fetch*, *Decode* and *Execute*. Give the values moved between or held in the affected components:
AC, PC, CI, MDR, MAR, AIU, EQ, GT, OV

Fetch:

Decode:

Execute:

[9 marks]

Question 4 [25 marks] More PeANUt

(a) The PeANUt program below consists of a main program and a function `doit`. This function has two input arguments, one local variable and it returns one value. The input arguments, the local variable and the return value are all 16-bit integer values.

In the comments to this code you find numbers <1> to <7>. Analyse the code and answer the questions below.

```

main:      incsp #1      ; <5>  main() {
          load #3
          incsp #1
          store !0
          load #6
          incsp #1
          store !0      ; <2>
          call doit
          incsp #...    ; <4>
          load !0
          incsp #-1
          add #48
          trap #3
          load #10     ; <7>
          trap #3     ; <7>
          trap #1

RV = ...
a = ...
b = ...
c = ...

doit:     <3>
          ; int doit(int a, int b) {
          ; int c;
          ;
          ; if (a > b) {
          ;   c = a+b;
          ; }
          ; else {
          ;   c = b - a;
          ; }
          ; return c;
          ; }
          end main
  
```

The following questions refer to the PeANUt assembly code above.

Give the values for the four concise macros at <1>.

a1: [2 marks]

Question 4 (continued)

Student Number:

How does the stack look (i) after the `store` instruction at <2> has been executed, and (ii) at the beginning of the procedure `doit` at <3>. Draw two diagrams of the stack, assuming the stack pointer (SP) had been pointing to address 100₈ at the beginning of the main program. Show the contents of the stack, and where the stack pointer points to in both cases.

a2: [6 marks]

By which value has the stack pointer to be changed at the instruction <4>?

a3:

What is the purpose of the instruction at <5>?

a4: [1 mark]

What is the purpose of the instruction at <6>?

a5: [1 mark]

Question 4 (continued)

What value will be stored in RV at the end of the procedure `doit`?

a6:

[1 mark]

What is the purpose of the two instructions at <7>?

a7:

[1 mark]

Program the function `doit` in PeANUit assembly code, assuming that the procedure convention as used by the main program is used, and that the input arguments have been pushed onto the stack.

a8:

[6 marks]

Question 4 (continued)

(b) What is the difference between exceptions and interrupts?

[3 marks]

(c) Describe the *Least Recently Used* (LRU) page replacement policy, which is often used in virtual memory systems.

[3 marks]

Student Number:

Question 5 [15 marks] Further Topics

- (a) Describe briefly how circuit switched, packet switched and broadcast networks differ. Give an example of each.

[3 marks]

- (b) The computer communication process is usually broken down into 3 (or more) layers. For example the application layer, transport layer and network layer. What is the rationale for having different layers? (NOTE - this question is asking WHY this is done, not what is done).

[2 marks]

Question 5 (continued)

Student Number:

- (c) PeANUt is an example of a program that simulates the operation of a basic computer system. PeANUt was written for teaching purposes. Give two other reasons why people may want to write programs that simulate the detailed workings of computers?

[2 marks]

- (d) What is a branch delay slot and how is it related to pipelining?

[2 marks]

Question 5 (continued)

(e) The following is a piece of SPARC assembly code.

```

/*
i=0;
j=0;
do{
    j=j+(i-7)*(i+8);
    i++;
}while(i<5);
*/
/*line01:*/      mov    0, %l0      ! %l0 = 0
/*line02:*/      mov    0, %l1      ! %l1 = 0
/*line03:*/      .top:    ! label
/*line04:*/      nop                    ! null operation
/*line05:*/      sub    %l0,7,%o0    ! %o0 = %l0 - 7
/*line06:*/      add    %l0,8,%o1    ! %o1 = %l0 + 8
/*line07:*/      call   .mul                ! call function .mul
/*line08:*/      nop                    ! null operation
/*line09:*/      add    %l1,%o0,%l1    ! %l1 = %l1 + %o0
/*line10:*/      add    %l0,1,%l0    ! %l0 = %l0 + 1
/*line11:*/      cmp    %l0,5        ! compare %l0 and 5
/*line12:*/      blt   .top        ! goto .top if %l0 < 5
/*line13:*/      nop                    ! null operation

```

It contains, as comments, the equivalent C code, unique line numbers for every line of assembly code, and an interpretation of each instruction. Registers begin with a % character and are either out (o) or local (l) registers.

Give the line numbers for ALL instructions that correspond to branch delay slots.

e1:

[1 mark]

How can the above assembly code be rearranged to remove all the null operations (nop instructions) while also producing identical results? Give your answer in terms of the line numbers of the original code, in an ordered series beginning with the first line to be executed.

e2:

[2 marks]

Question 5 (continued)

(f) You are working on a parallel computer with identical processors. You have parallelised your code such that on a single processor 90% of the execution time is now spent in parts of the code that have been parallelised. Assuming your code obeys Amdahl's law, what would be the minimum number of processors you would need to use in order to achieve a speedup of AT LEAST 4 (i.e. ≥ 4). Show how you derive your answer.

[3 marks]

Student Number:

Continuation of answer to Question Part

Continuation of answer to Question Part

Continuation of answer to Question Part

Continuation of answer to Question Part

Appendix

formal	mode	opcode	name	meaning
One	v	000	—	illegal instruction
One	v'	001	Load	AC := OP
One	v'	010	Store	memory[OP] := AC
One	v	011	Addition	AC := AC + OP
One	v	100	Subtraction	AC := AC - OP
One	v	101	Division	AC := AC / OP
One	v	110	Multiplication	AC := AC * OP
One	v	111	Compare	compare AC to OP, set CCs
Two	f_i	101000	Jump	jump to address OP
Two	f_i	101001	BranchEqual	branch to AOP if EQ=1
Two	f_i	101010	BranchNotEqual	branch to AOP if EQ=0
Two	f_i	101011	BranchGreater	branch to AOP if GT=1
Two	f_i	101100	BranchLessEqual	branch to AOP if GT=0
Two	f_i	101101	BranchOverflow	branch to AOP if OV=0
Two	f_i	101110	LogicalAnd	AC := AC \wedge OP, bitwise
Two	f_i	101111	LogicalOr	AC := AC \vee OP, bitwise
Two	f_i	110000	LogicalXor	AC := AC \oplus OP, bitwise
Two	f_0	110001	SetIndexReg	XR := OP
Two	f_0	110010	IncIndexReg	XR := XR + OP
Two	f_0	110011	IncStackPoint	SP := SP + OP
Two	f_i	110100	CallProcedure	call procedure at OP
Two	f_0	110101	Trap	perform trap number OP
Two	f_i	110110	LoadAddress	AC := AOP
Three	—	1110000	Return	procedure or interrupt return
Three	—	1110001	ClearOver	OV := 0
Three	—	1110010	LoadPSW	AC := PSW
Three	—	1110011	StorePSW	PSW[13..10] := AC[13..10]
Three	—	1110100	LogicalNot	AC := \neg AC, bitwise
Three	—	1110101	CompareIndexReg	compare XR to AC, set CCs
Three	—	1110110	LoadIndexReg	AC := XR
Three	—	1110111	StoreIndexReg	XR := AC
Three	—	1111000	LoadStackPoint	AC := SP
Three	—	1111001	StoreStackPoint	SP := AC

Table 1: PeANut machine language instructions

code	modes	comments
v	any mode (0-4)	must be explicitly specified (3 bits)
v'	any mode except 0	must be explicitly specified (3 bits)
f_0	mode 0 (fixed)	implicitly specified by opcode
f_1	mode 1 (fixed)	implicitly specified by opcode
—	no mode is applicable	

Table 2: Addressing modes as listed in Table 1

machine instruction	operation	machine instruction	operation
Load	load	SetIndexReg	setxr
Store	store	IncIndexReg	incxr
Addition	add	IncStackPoint	incsp
Subtraction	sub	CallProcedure	call
Division	dvd	Trap	trap
Multiplication	mul	LoadAddress	loada
Compare	cmp	Return	ret
Jump	jmp	ClearOver	clov
BranchEqual	beq	LoadPSW	ldpsw
BranchNotEqual	bne	StorePSW	stpsw
BranchGreater	bgt	LogicalNot	not
BranchLessEqual	ble	CompareIndexReg	cmpxr
BranchOverflow	bov	LoadIndexReg	loadxr
LogicalAnd	and	StoreIndexReg	storexr
LogicalOr	or	LoadStackPoint	loadsp
LogicalXor	xor	StoreStackPoint	storesp

Table 3: Instruction Operations

x	2^x
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024
11	2048
12	4096
13	8192
14	16384
15	32768

Table 4: Powers of 2 in decimal