

THE AUSTRALIAN NATIONAL UNIVERSITY
First Semester Examination – June 2003

COMP2300
Introduction to Computer Systems

Study Period: 15 minutes

Time Allowed: 3 hours

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

NO calculator permitted.

Questions are NOT equally weighted.

The questions are followed by labelled, framed blank panels 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:

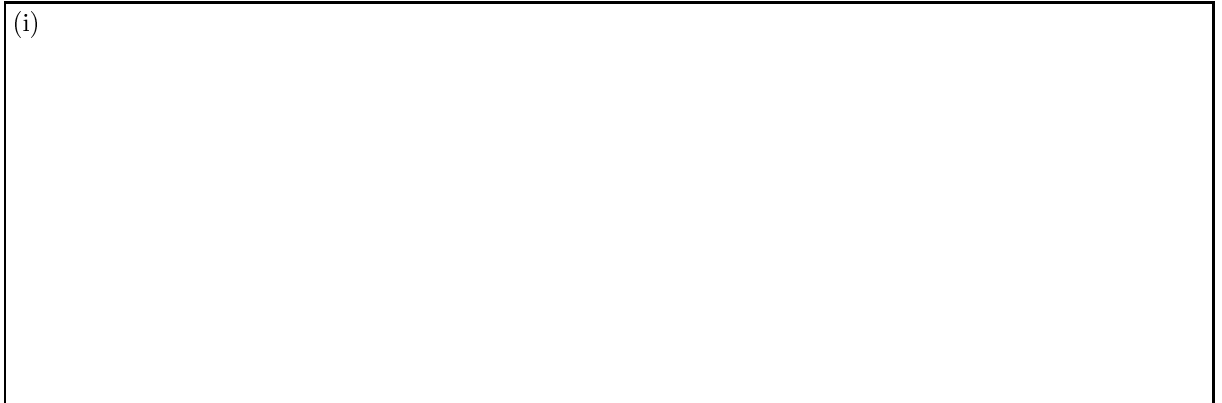
The following are for use by the examiners.

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

Question 1 [15 marks] Fundamental Concepts

- (a) (i) Stack architectures are one class of instruction set architectures. Explain what is meant by stack architecture.

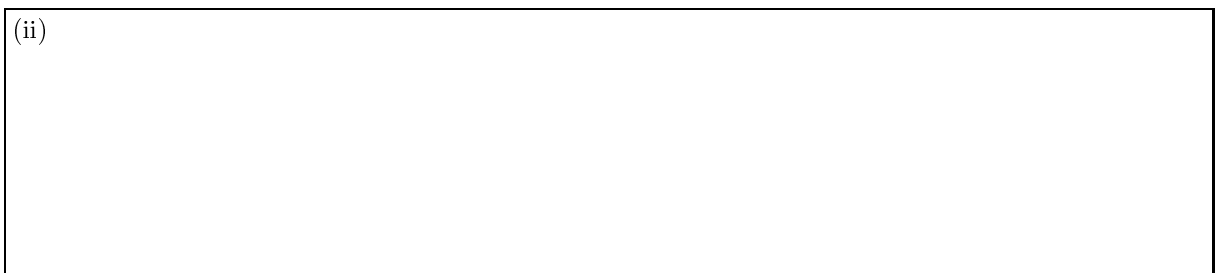
(i)



[2 marks]

- (ii) Is PeANUt a stack architecture? Justify your answer (failure to do so will result in no marks).

(ii)



[1 mark]

- (b) Convert the base 9 value 2345_9 to *hexadecimal*. Show your workings.



[3 marks]

Question 1 (continued)

- (c) 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 floating point value is represented by the following bit pattern?

11000001 01010100 00000000 00000000

[3 marks]

- (d)

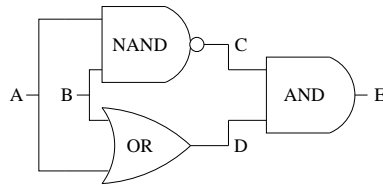
$$\begin{aligned}
 X &= -37 \text{ (decimal integer)} \\
 Y &= 1110\ 1101\ 1010\ 0111 \text{ (16 bit two's complement integer)} \\
 Z &= X - Y
 \end{aligned}$$

In the above what is the value of Z. Give your answer as both a 16 bit two's complement binary number AND as a decimal number. Show your working.

[4 marks]

Question 1 (continued)

- (e) The following circuit is constructed using AND, OR and NAND gates. 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.



- (i) For the following input values of A and B what are the output values at points C, D and E?

(i)

A	B	C	D	E
0	0			
0	1			
1	0			
1	1			

[1 mark]

- (ii) What is the name of the binary operator that for the bit operands given in columns A and B gives the same result as column E above?

(ii)

[1 mark]

Question 2 [20 marks] C Programming Language

(a) The following program examines two character strings using functions from the standard C library:

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

int main() {

    char name1 []="My COMP2300\notes";
    char name2 []="Year 2002          ";

    int i, ndigit=3, nalpha=0;

    printf( /* "Title\n");
    printf( /* "%s:%s\n",name1,name2);

    for (i=0;i<15;i++){
        if (isdigit((int) name1[i]) || isdigit((int) name2[i])) ndigit++;
        if (isalpha((int) name1[i]) && isalpha((int) name2[i])) nalpha++;
    }

    printf("Total number of digits      = %4d\n",ndigit);
    printf("Total number of alphabetic = %4d\n",nalpha);

    return 0;
}
```

Relevant details from the `isalpha/isdigit` man page are given below:

SYNOPSIS

```
#include <ctype.h>

int isalpha (int c);
int isdigit (int c);
```

DESCRIPTION

```
isalpha()
    checks for an alphabetic character [a-zA-Z]; it is
    equivalent to (isupper(c) || islower(c)).
isdigit()
    checks for a digit (0 through 9).
```

RETURN VALUE

The values returned are nonzero if the character `c` falls into the tested class, and a zero value if not.

Question 2 (continued)

(i) 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.

(i)

[4 marks]

(ii) Write C code for a function

```
int myisdigit(int c);
```

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

(ii)

[3 marks]

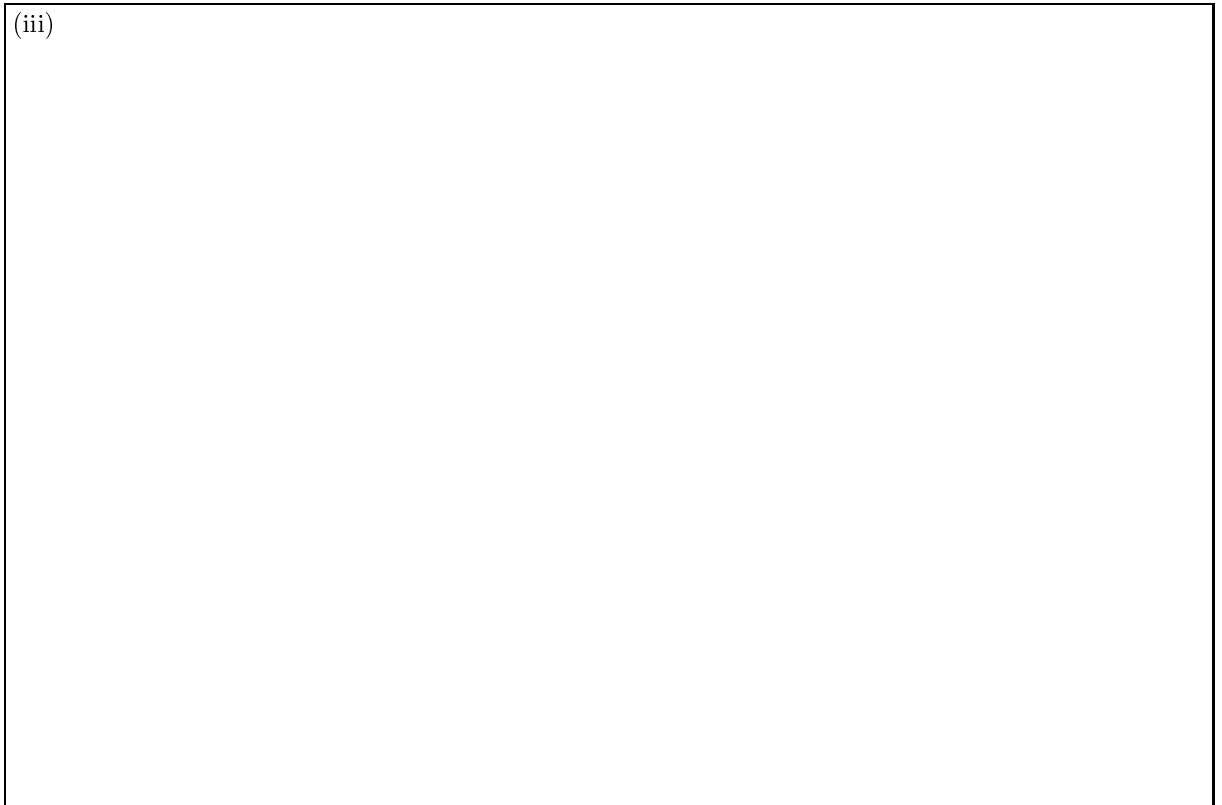
Question 2 (continued)

(iii) Write C code for a function

```
int myisalpha(int c);
```

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

(iii)



[4 marks]

Question 2 (continued)

- (b) The following gives two alternative versions of the function `convert`. The function is supposed to take an integer as input and print its value using any base up to and including base 16. The base to be used is also given as input to `convert`. `convert1` is incorrect, while `convert2` is correct.

```

void convert1(int *vl, int bs) {
    int i,ii;
    char cde[16]="0123456789ABCDEF";
    char final[64];

    ii=0;
    while(vl > 0){
        vl = vl/bs;
        final[ii]=cde[vl%bs];
        ii++;
    }
    printf("%8d in base %2d = ",vl,bs);
    for (i=0;i<ii;i++)
        printf("%c",final[i]);
    printf("\n");
}

void convert2(int vl, int bs) {
    int i,ii;
    char cde[]="0123456789ABCDEFHIJ";
    char final[64];

    printf("%8d in base %2d = ",vl,bs);
    ii=0;
    while(vl > 0){
        final[ii]=cde[vl%bs];
        vl /= bs;
        ii++;
    }
    for (i=ii-1;i>=0;i--)
        printf("%c",final[i]);
    printf("\n");
}

```

- (i) List THREE fundamentally different problems with `convert1`.

(i)

[3 marks]

- (ii) We wish to call `convert2` from the following program:

```

int main( ) {
    int val[5];
    val[0]=13; val[1]=435; val[2]=1293; val[3]=8383; val[4]=19;

    /* call to function convert HERE */

    return 0;
}

```

Detail the LINES of code you would add to function `main` in order to call `convert2` and print out the hexadecimal representation of ALL 5 values in the `val` array?

(ii)

[2 marks]

Question 2 (continued)

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

```
#include <stdio.h>
int mystery(char *val);

int main() {
    char name1[]="Computer Systems Examination";
    int n;

    printf("Mystery Question\n");
    n = mystery(&name1[10]);
    printf("name1=%s:n=%d\n",name1,n);

    return 0;
}

int mystery(char *val) {
    int xyz=0;
    char *pt1,*pt2,pt3;

    pt2=val;
    while (*pt2 != '\0') {
        xyz++;
        pt2++;
    }
    pt2--;
    pt1=val;
    while (pt2 > pt1) {
        pt3=*pt1;
        *pt1+=*pt2;
        *pt2--=*pt3;
    }
    return xyz;
}
```

(i) Explain briefly what function `mystery` is doing

(i)

[2 marks]

(ii) Exactly what is printed out when the program is run?

(ii)

[2 marks]

Question 3 [25 marks] PeANUt

- (a) Assume you can choose to develop a program using either PeANUt machine language instructions or PeANUt assembly language. Give three reasons why you would select one above the other.

[3 marks]

- (b) The PeANUt architecture contains a register called **SP**. Explain in two sentences what this register is and what it is used for.

[2 marks]

- (c) (i) Explain in two to three sentences what the following PeANUt assembly code is doing. Assume that an array of length 10 of integer numbers (with values between 0 and 9) is stored from **a10** onwards.

```
load #4
storexr
load *a10
add #'0'
trap #3
```

(i)

[2 marks]

- (ii) Which addresses in memory are accessed by the above PeANUt code? Write down the addresses in decimal, and if they are read or write accesses.

(ii)

[1 mark]

Question 3 (continued)

- (d) Explain how procedure calls are facilitated by the PeANUt machine when you use the **call** and **ret** instructions.

[2 marks]

- (e) Assume the accumulator **AC** contains a value of **7** (decimal), and at address **a3** the value **6** (decimal) is stored. The memory cell at address **a6** contains a value of **5**.

- (i) Explain in detail what happens when the PeANUt machine executes the instruction **cmp @3**. This instruction (**cmp @3**) is stored at address **a20**. Your answer should describe the events that happen in the phases *Fetch*, *Decode* and *Execute* and the values moved between the affected components **AC**, **CI**, **MAR**, **MDR** and **ALU**.

(i)

[4 marks]

- (ii) Write down the values (in decimal) stored in the **AC**, the **MAR**, the **MDR**, and the flags **EQ**, **GT** and **OV** after the instruction **cmp @3** has been executed.

(ii)

[2 marks]

Question 3 (continued)

- (f) (i) Write a sequence of PeANUt assembly language instructions that do the same as the following piece of C code. Use a **trap #1** instruction to end your program.

```
int i = 0;
int sum = 0;
int end = 10;
int a[10];

for (i = 0; i < end; i++) {
    a[i] = sum + 1;
    sum = sum + a[i];
}
```

(i)

[8 marks]

- (ii) Write down the value of the integer **sum** after the program has been executed.

(ii)

[1 mark]

Question 4 [25 marks] More PeANUt

(a) The PeANUt program below consists of a main program and a function `myfunct`. For the following questions we assume that the procedure call convention as discussed in the COMP2300 lectures is used. On the right side you see the stack, shown with content at the beginning of the function `myfunct` (just after the function has been called but before the first instruction in it has been executed), and the position of the stack pointer at the beginning of `myfunct`.

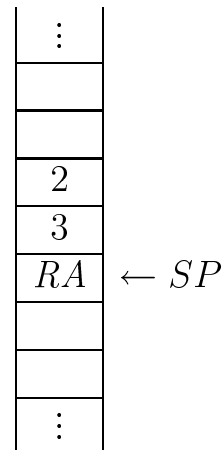
In the comments to this code and with the stack you find numbers in brackets `<1>` to `<14>`. Analyse the code and the stack and answer the questions below.

```

main:
    incsp #1
    load # ...    ; <1>
    incsp #1
    store !0
    load # ...    ; <2>
    incsp #1
    store !0
    call myfunct
    ...           ; <3>
    ...           ; <4>
    ...           ; <5>
    add #'0'
    trap #3
    load #10      <11> ...
    trap #3      <12> ...
    trap #1      <13> ...

    RV = ...     ; <6>
    a = ...      ; <7>
    b = -2
    t = ...      ; <8>
    NumLocs = 1

myfunct:
    ...           ; /* SP position when here */
    ...           ; <9>
    load !a
    mul !b
    store !t
    add !t
    div !b
    store !RV
    ...           ; <10>
    ret
    
```



(i) Complete the load instructions at `<1>` and `<2>` with the specific values of their arguments.

(i)

[1 mark]

Question 4 (continued)

(ii) Write down the three instructions that are needed at positions <3>, <4> and <5> so that the value returned from the function is loaded into the accumulator, and after these three instructions have been executed the stack pointer is pointing to the same memory cell as it would have been at the beginning of the main program. Follow the procedure call convention.

(ii)

[2 marks]

(iii) What are the values of the offsets for **RV**, **a** and **t** at positions <6>, <7> and <8>?

(iii)

[1 mark]

(iv) Write down the two instructions that are needed at positions <9> and <10> in order to follow the procedure call convention.

(iv)

[1 mark]

(v) Label the four memory cells on the stack at positions <11> to <14>.

(v)

[1 mark]

(vi) What value is stored in the return value **RV** at the end of the function **myfunct**? And what value is stored in the variable **t**?

(vi)

[1 mark]

Question 4 (continued)

(vii) Write **C** code for the function `myfunct` that does the same as the assembly code given on page 13.

(vii)

[3 marks]

(b) Assume you have a multi-process operating system – like Unix as running on the student servers.

(i) Why is *time slicing* important in multi-processing? What can happen if no time slicing is implemented? Explain briefly.

(i)

[2 marks]

(ii) What information needs to be stored when a running process is removed from the CPU and inserted into the *ready queue*?

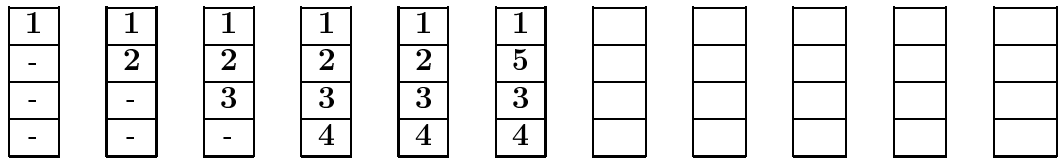
(ii)

[2 marks]

Question 4 (continued)

(c) Given a virtual memory system with a main memory that can hold four memory pages. Assume you have the following sequence of 11 page accesses at times t_1 to t_{11} to 5 different pages (numbered 1 to 5):

Time:	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}
Page:	1	2	3	4	1	5	2	3	2	4	1

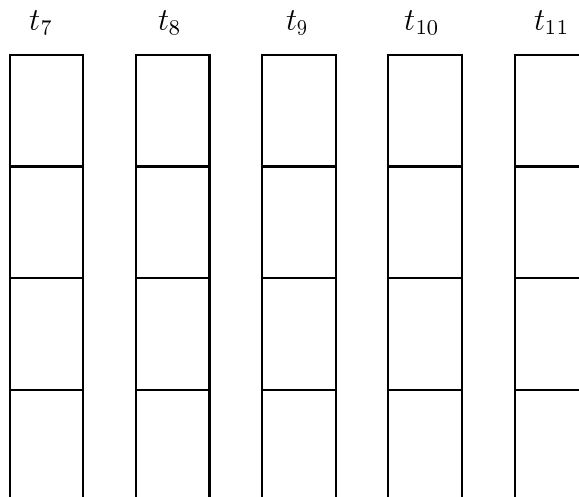


(i) Which page replacement policy is used? Explain in one to two sentences how this policy works.

(i)

[2 marks]

(ii) Complete the above diagram for the times t_7 to t_{11} . Please write your answers into the following boxes.



[3 marks]

Question 4 (continued)

(iii) Give two main reasons why virtual memory is implemented on almost all modern computer systems.

(iii)

[2 marks]

(d) Explain the differences between traps and procedures as used in PeANUt. Give one example where you would implement a trap rather than a procedure.

[2 marks]

(e) Explain in two to three sentences how files are stored and accessed on a hard disk.

[2 marks]

Question 5 [15 marks] Further Topics

- (a) Bandwidth and latency are two important characteristics of communication networks. Define what you think is meant by each of these terms. Give an example of a situation when communication latency is critical and another example of when communication bandwidth is more important. Justify your examples.

[3 marks]

- (b) Grid services are a major driving force in the development of advanced network technologies. Give two examples of grid services and explain what they are.

[2 marks]

Question 5 (continued)

- (c) In computer communications long messages are broken down into packets. Each packet is given a header. List **THREE** different pieces of information that are included in typical packet headers, **AND** give the purpose of each.

[3 marks]

- (d) The Java Virtual Machine (JVM) defines a region of memory called the *Constant Pool*. What is the Constant Pool **AND** what possible advantage is there in having this as a special region of memory?

[2 marks]

Question 5 (continued)

(e) The following piece of C code

```

i=0;
j=0;
do {
    j=j+(i-7)*(i+8);
    i++;
} while (i<5);

```

can be evaluated using the following lines of SPARC assembly.

```

/*line01:*/ .top:          ! label
/*line02:*/      call      .mul          ! call function .mul
/*line03:*/      blt      .top          ! goto .top if %10 < 5
/*line04:*/      cmp      %10,5        ! compare %10 and 5
/*line05:*/      mov      0, %10        ! %10 = 0
/*line06:*/      mov      0, %11        ! %11 = 0
/*line07:*/      sub      %10,7,%o0    ! %o0 = %10 - 7
/*line08:*/      add      %10,8,%o1    ! %o1 = %10 + 8
/*line09:*/      add      %10,1,%10    ! %10 = %10 + 1
/*line10:*/      add      %11,%o0,%11  ! %11 = %11 + %o0

```

Unfortunately the lines have become jumbled and are no longer in the correct order. Reorder the lines so that they evaluate the C code correctly. Give your answer in terms of the line numbers given, e.g. 10,3,6,... Note, each line is used just once and no other lines (or no-ops) are required.

[2 marks]

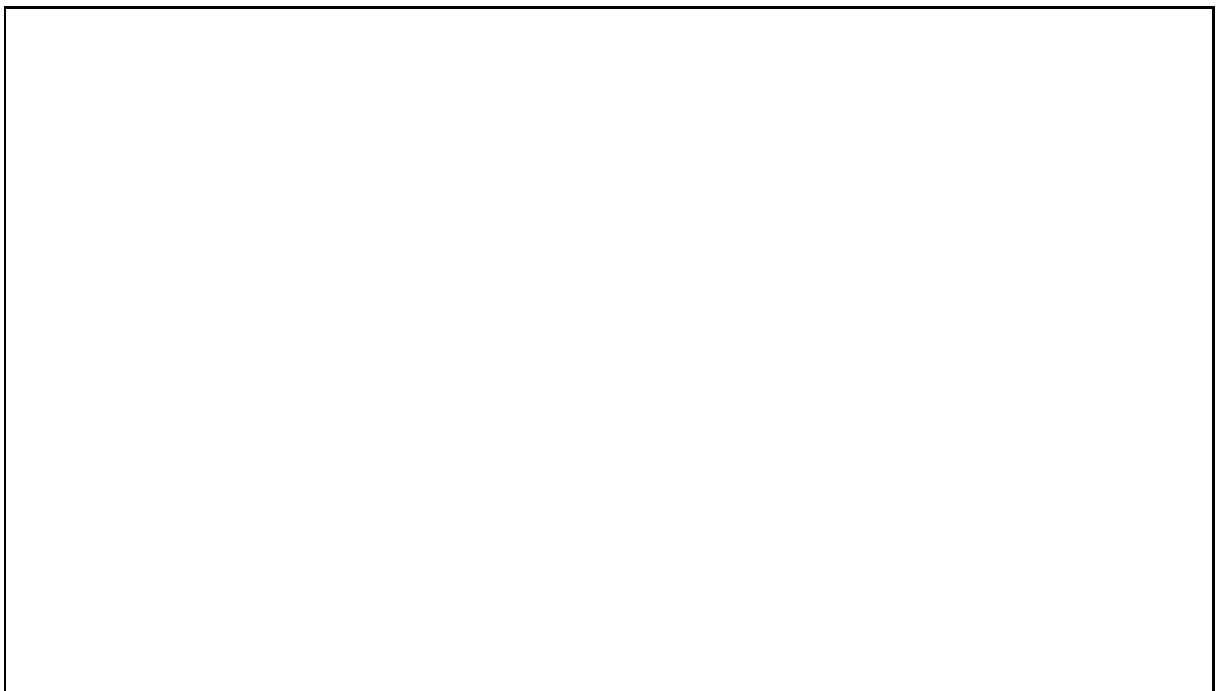
Question 5 (continued)

(f) All modern memory architectures include various caches. What is the purpose of these caches?



[1 mark]

(g) Your parallel code takes 110 seconds to complete using 2 processors and 51 seconds on 6 processors. Estimate how long it would take using 10 processors. Provide a detailed explanation of how you derive your estimate.



[2 marks]

Student Number:

Continuation of answer to Question Part

Continuation of answer to Question Part

Student Number:

Continuation of answer to Question Part

Continuation of answer to Question Part

Appendix

<i>format</i>	<i>mode</i>	<i>opcode</i>	<i>name</i>	<i>meaning</i>
One	<i>v</i>	000	—	illegal instruction
One	<i>v</i>	001	Load	AC := OP
One	<i>v'</i>	010	Store	memory[AOP] := AC
One	<i>v</i>	011	Addition	AC := AC + OP
One	<i>v</i>	100	Subtraction	AC := AC - OP
One	<i>v</i>	101	Division	AC := AC / OP
One	<i>v</i>	110	Multiplication	AC := AC * OP
One	<i>v</i>	111	Compare	compare AC to OP, set CCs
Two	<i>f₁</i>	101000	Jump	jump to address AOP
Two	<i>f₁</i>	101001	BranchEqual	branch to AOP if EQ=1
Two	<i>f₁</i>	101010	BranchNotEqual	branch to AOP if EQ=0
Two	<i>f₁</i>	101011	BranchGreater	branch to AOP if GT=1
Two	<i>f₁</i>	101100	BranchLessEqual	branch to AOP if GT=0
Two	<i>f₁</i>	101101	BranchOverflow	branch to AOP if OV=0
Two	<i>f₁</i>	101110	LogicalAnd	AC := AC ∧ OP, bitwise
Two	<i>f₁</i>	101111	LogicalOr	AC := AC ∨ OP, bitwise
Two	<i>f₁</i>	110000	LogicalXor	AC := AC ⊕ OP, bitwise
Two	<i>f₀</i>	110001	SetIndexReg	XR := OP
Two	<i>f₀</i>	110010	IncIndexReg	XR := XR + OP
Two	<i>f₀</i>	110011	IncStackPoint	SP := SP + OP
Two	<i>f₁</i>	110100	CallProcedure	call procedure at OP
Two	<i>f₀</i>	110101	Trap	perform trap number OP
Two	<i>f₁</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 := ¬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

<i>machine instruction</i>	<i>operation</i>	<i>machine instruction</i>	<i>operation</i>
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