

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 answers in a clear manner than to outline a greater number in a sketchy, half-answered fashion.

Please write clearly – if we can not read your writing you might lose marks!

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 [10 marks] Digital Building Blocks

(a) Memory addresses 0x02AD4C39 - 0x02AD4C3D hold the following 8-bit binary values:

Address	0x02AD4C39	0x02AD4C3A	0x02AD4C3B	0x02AD4C3C	0x02AD4C3D
Binary Value	01000010	10010101	01001000	00101101	11011111

(i) If `sizeof(x)=2` and `&x=0x02AD4C3A` what would be printed by the statement

```
printf("Unsigned value of X = %u\n",x)
```

if the data storage were big endian?

(i)

[2 marks]

(ii) If `sizeof(y)=2` and `&y=0x02AD4C3C` what would be printed from the statement

```
printf("Value of Y = %d\n",y)
```

if the data storage were little endian?

(ii)

[2 marks]

Question 1 (continued)

Student Number:

- (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. To **1 decimal place** what floating point number is represented by the following bit pattern

01000010 10010101 00000000 00001101

[2 marks]

(c)

$X = 1100\ 1111$ (8 bit two's complement integer)
 $Y = 1111\ 1100\ 1110\ 0110$ (16 bit two's complement integer)
 $Z = X - Y$

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.

[2 marks]

Question 1 (continued)

Student Number:

- (d) If w , x , y , and z are variables stored in memory locations M_w , M_x , M_y and M_z respectively, sketch how the expression $z = y + x * w$ might be evaluated on a load/store architecture. Assume that you have access to 8 registers denoted R_0 - R_7 . Clearly define the meaning of the instructions you use.

[2 marks]

Question 2 [15 marks] C Programming

- (a) The standard C library contains a function called `atoi` that converts a base 10 integer value encoded as a string of ASCII digits (i.e. ASCII characters for '0' through '9') to the equivalent integer value stored as an `int`. You have written your own version of `atoi` that is given below:

```
int my_atoi(char str[])
{
    int value=0, i=0;

    while ( str[i]!='\0' ){
        if ( str[i] >= '0' && str[i] <= '9' ){
            value*=10;
            value+=( (int)str[i] - (int)'0' );
        }
        i++;
    }

    return value;
}
```

The routine compiles without error.

- (i) For some input character strings `my_atoi` will perform the translation correctly, giving the expected return value. Give one example of an input character string for which this is the case. (Give your answer as `str []="something"`)

(i)

[1 mark]

- (ii) For other input character strings `my_atoi` will perform the translation incorrectly, despite the fact that the character string contains only the ASCII characters for digits 0-9 and the null character. Give one example of such a character string and explain precisely why the result will be incorrect.

(ii)

[1 mark]

Question 2 (continued)

To test `my_atoi` you write the following `main` program:

```
#include <stdio.h>
int my_atoi(char *);
int main(void)
{
    char input[20];
    int value,total=0;

    printf("\nInput data\n");
    do {
        scanf("%s",input);
        value=my_atoi(input);
        total+=value;
        printf("Value %5.5d Total %-5d\n",value, total);
    } while (total < 1000);

    return 0;
}
```

- (iii) The routines compile and link without problems. You run the resulting executable and provide the following input:

```
1
+2 -3
4.5 1.2E+01 10000
```

Exactly what output will you obtain? Clearly indicate all white space using '␣' for each space.

(iii)

[3 marks]

Question 2 (continued)

Student Number:

(iv) The above executable is liable to buffer overflow errors. What is a “buffer overflow error”, how can it occur in the above code, and how would you modify the code in order to prevent buffer overflow errors?

(iv)

[3 marks]

Question 2 (continued)

Student Number:

(v) Re-write `my_atoi` so that it can be used to convert a hexadecimal integer encoded as an ASCII character string to its equivalent base 10 value stored as an `int`. You should assume that the input character string is of the form `0x*****` where `0x` has been used to indicate that the encoded integer is hexadecimal and the `*` are used to indicate AT MOST 8 ASCII characters each of which can be any of `0-9`, `a-f`, or `A-F`. Your routine must NOT call any other C library function.

(v)

[4 marks]

Question 2 (continued)

Student Number:

(b) A C program contains the following declaration:

```
static int array[2][3]={33,16,29,54,87,99};
```

(i) What are the meanings of (array+1) and *(array+1)+1?

(i)

[1 mark]

(ii) What are the values of *((array+1)+1) and *(*(array+1))+1?

(ii)

[1 mark]

(iii) What is the value of (array[1][2]-(&array[0][0]+4))*((int)(array+1)-(int)array)? Show how you derive your answer.

(iii)

[1 mark]

Question 3 [25 marks] Assembly Level Machine Organisation

Student Number:

(a) (i) Which of the following six assembly instructions is not a valid PeANUt instruction? Explain in one sentence.

- A: store @1023
- B: xor -1
- C: bov a42
- D: ldpsw 0
- E: setxr #-3
- F: clouv

(i)

[1 mark]

(ii) What kind of error message would you expect the PeANUt assembler to print if you have the above non-valid instruction in your program?

(ii)

[1 mark]

(b) (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 is stored from a10 (octal) onwards (containing values between 0 and 9, i.e. a10 contains 0, a11 contains 1, and so on). What character is printed?

```
loada a10
storexr
load *9
add #'0'
trap #3
```

(i)

[2 marks]

(ii) Which address(es) in memory is (or are) accessed by the above PeANUt code? Write down the address(es) in octal, and if it (or they) are read or write access(es).

(ii)

[1 mark]

Question 3 (continued)

Student Number:

- (c) The PeANUt architecture supports an *indexed* addressing mode. Explain in one to two sentences how this addressing mode works, for what it is used, and how it is supported by the PeANUt architecture.

[1 mark]

- (d) Assume the accumulator **AC** contains a value of **511** (decimal), and at address **a1** (octal) the value **2** (decimal) is stored. The memory cell at address **a2** (octal) contains a value of **256**.

(i) Explain in detail what happens when the PeANUt machine executes the instruction **mul @1**. Assume this instruction (**mul @1**) is stored at address **a10**. 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 **PC**, the **MAR**, the **MDR**, and the flags **EQ**, **GT** and **OV** after the instruction **mul @1** has been executed.

(ii)

[1 mark]

Question 3 (continued)

Student Number:

- (e) (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 a = 0;
int b = 1;
int c = 0;
int d[10];
int e = 10;

while (c < e) {
  d[c] = a + b;
  a = b;
  b = d[c];
  c = c + 1;
}
```

(i)

[6 marks]

(ii) Write down the values in all elements of the array **d** after the above program has been executed.

(ii)

[1 mark]

Question 3 (continued)

(f) Assume you have a PeANUt assembly program that consists of a main part and a function `myfunct` as shown below. The function `myfunct` is recursive, which means it calls itself. Analyse the given assembly code and answer the questions (i) to (v) on the following pages.

In the comments to this code you find three numbers in brackets (<1>, <2> and <3>), which will be referred to in question (iii) on the next page.

On the right side you see parts of the PeANUt memory. The *stack pointer (SP)* is shown pointing to the memory cell with address `a100` at the beginning of the program (i.e. after the PeANUt has been initialised and the program has been loaded, but before it has been started).

```

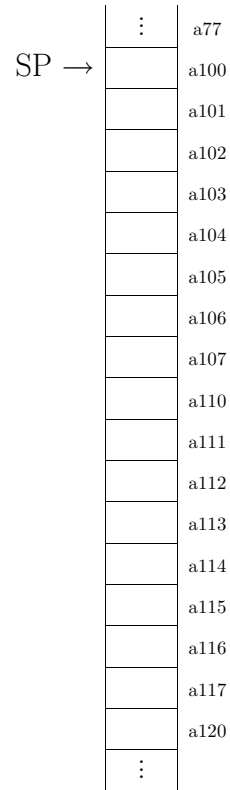
MAXREC = 3
x:      data      1

RV =    -3
i =     -2
t =      0
NumLocs = 1

myfunct: incsp    #NumLocs
         load     !i
         cmp     #MAXREC
         bne    myelse
         store   !t          ; <1>
         jmp     myend
myelse:  add     #1
         incsp   #1
         incsp   #1
         store   !0
         call   myfunct
         incsp   #-1
         load    !0
         incsp   #-1
         store   !t
myend:   load    !t
         store   !RV
         incsp   #-NumLocs
         ret

main:    incsp    #1
         incsp    #1
         load     x
         store   !0          ; <2>
         call   myfunct
         incsp   #-1          ; <3>
         load    !0
         incsp   #-1
         add     #'0'
         trap    #3
         load    #'\\n'
         trap    #3
         trap    #1

end      main
    
```



Question 3 (continued)

The following questions refer to the PeANUt assembly code shown on the previous page.

(i) Both `x` and `t` are integers. What type of variable is `x`, what is `t`?

(i)

[1 mark]

(ii) Fill-in the stack diagram on the right side of the previous page when the program is executed. Use `RA1`, `RA2`, etc. for return addresses; and `RV1`, `RV2`, etc. for return values. Fill in numerical values if they are available (for example for values of `i` and `t`).

Write your answers directly into the stack diagram on the previous page!

[2 marks]

(iii) To which memory cells is the *stack pointer (SP)* pointing after the instructions at <1>, <2> and <3> have been executed? Please write down the octal address values in the form $SP_1 = a\dots$, $SP_2 = a\dots$ and $SP_3 = a\dots$.

(iii)

[1 mark]

(iv) Write C code for the main program that does the same as the `main` PeANUt code shown on the previous page. Use a `printf()` for the `trap #3` instructions.

(iv)

[1 mark]

Question 3 (continued)

Student Number:

(v) Write C code for the function `myfunct` that does the same as the PeANUt code `myfunct` shown on page ??.

(v)

[1 mark]

(g) Assume the PeANUt accumulator contains the bit pattern **01010101 10101010** and at address **a10** (octal) in main memory the decimal value **126** is stored. The following two instructions are executed:

```
not
xor a10
```

What binary value (16 bits) is stored in the accumulator after these two instructions have been executed?

[1 mark]

Question 4 [10 marks] Memory Systems and Modern Machines

Student Number:

(a) 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:	4	3	2	1	4	5	4	3	2	5	1

4	4	4	4	4	5					
-	3	3	3	3	3					
-	-	2	2	2	2					
-	-	-	1	1	1					

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

(i)

[1 mark]

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

	t_7	t_8	t_9	t_{10}	t_{11}

[2 marks]

Question 4 (continued)

- (b) Imagine you have a Linux desktop machine with 512 Megabytes of main memory, and a program with a *working set* of 1 Gigabyte. Will it be possible that this program runs with good paging behaviour (i.e. execute with a small number of page faults)? If so, explain how. If the program will never exhibit good paging behaviour explain why.

[2 marks]

- (c) One characteristics of RISC machines is the use of fixed length instructions. Give one advantage and one disadvantage of using fixed length instructions.

[1 mark]

- (d) In lectures it was stated that for a superscalar processor to perform well, the code must have the right instruction mix. What does this mean?

[1 mark]

Question 4 (continued)

- (e) The following shows three functions (`clear1`, `clear2`, and `clear3`) that perform the same operations with varying degrees of spatial locality:

```

void clear2(point *p, int n)
{
    int i,j;
    for (i = 0; i < n; i++){
        for (j = 0; j < 3; j++){
            p[i].vel[j] = 0;
            p[i].acc[j] = 0;
        }
    }
}

#define N 1000
typedef struct{
    int vel[3];
    int acc[3];
}point;

point p[N];

void clear1(point *p, int n)
{
    int i,j;
    for (i = 0; i < n; i++){
        for (j = 0; j < 3; j++){
            p[i].vel[j] = 0;
            for (j = 0; j < 3; j++){
                p[i].acc[j] = 0;
            }
        }
    }
}

void clear3(point *p, int n)
{
    int i,j;
    for (j = 0; j < 3; j++){
        for (i = 0; i < n; i++){
            p[i].vel[j] = 0;
            for (i = 0; i < n; i++){
                p[i].acc[j] = 0;
            }
        }
    }
}
    
```

Each function accesses all elements of `p[N]`, although the order in which these accesses take place differs from function to function. For each function discuss the spatial locality of the accesses to `p[N]`. Which function would you expect to perform best, and which function would you expect to perform worst?

[3 marks]

Question 5 [10 marks] Operating System Concepts and Interconnection Networks

(a) Why are memory page sizes always powers of 2?

[1 mark]

(b) Describe the actions taken by the kernel to context-switch between processes.

[1 mark]

(c) You purchase a new 10000RPM disk drive for your computer. The documentation states that the average seek time is 5ms and that on average there are 500 sectors per track.

(i) Estimate the best possible sustained transfer rate (in Megabytes/sec) that you might expect to see from this disk.

[2 marks]

Question 5 (continued)

(ii) Outline the circumstances that will give rise to the worse possible transfer rate from this disk.

[1 mark]

(d) What are system calls? Give two examples of system calls.

[2 marks]

Question 5 (continued)

Student Number:

Student Number:

- (e) Is it always crucial to know that the message you have sent over a communication network has arrived at its destination safely? If your answer is "yes", explain why. If your answer is "no", give appropriate examples.

[1 mark]

- (f) In relation to the transmission of data over a communication network, give one advantage and one disadvantage of Manchester encoding over non-return-to-zero encoding.

[1 mark]

- (g) Long messages sent over TCP/IP are broken into smaller packets. Give two advantages that arise from breaking long messages into small packets.

[1 mark]

Continuation of answer to Question Part

Continuation of answer to Question Part

Student Number:

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

Continuation of answer to Question Part

Continuation of answer to Question Part

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 <code>opcode</code>
f_1	mode 1 (fixed)	implicitly specified by <code>opcode</code>
-	no mode is applicable	

Table 2: Addressing modes as listed in Table ??

<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