

THE AUSTRALIAN NATIONAL UNIVERSITY  
First Semester Examination – June 2006

**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 refers to.*

*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 cannot read your writing you might lose marks!***

*The Appendix contains information on the PeANUt instruction set, as well as a table with powers of 2 values in decimal.*

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) Assume memory addresses 0x02DD1411 to 0x02DD1415 contain the following 8-bit binary values:

Address	0x02DD1411	0x02DD1412	0x02DD1413	0x02DD1414	0x02DD1415
Binary value	01000010	10010101	11001000	11010011	00100000

Assume `sizeof(x) = 2` and `sizeof(y) = 2`, `&x = 0x02DD1412` and `&y = 0x02DD1414`, and the data storage is *little endian*.

(i) What would be printed by the following C statement?

```
printf("Values for x+y: %x %o", x+y, x+y);
```

Clearly show how you derive your answers.

(i)

[2 marks]

(ii) What would be printed by the following C statement?

```
printf("Values for x+y: %d %u", x+y, x+y);
```

Clearly show how you derive your answers.

(ii)

[2 marks]

**Question 1 (continued)**

- (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 **three decimal digits** what floating point number is represented by the following 32-bit number (given in hexadecimal representation):

0x40C40000

[2 marks]

- (c) What are the five main components of every computer architecture?

[1 mark]

Question 1 (continued)

(d) Is PeANU<sub>t</sub> a *load/store* architecture? Explain in one or two sentences.

[1 mark]

$$\begin{aligned} X &= -45 \text{ (decimal integer)} \\ Y &= 11001010 \text{ (8 bit two's complement integer)} \\ Z &= X - Y \end{aligned}$$

(e) In the above what is the value of Z? Give your answer as **both** an 8-bit two's complement binary number **and** as a decimal number. Show your working.

[2 marks]

**Question 2 [15 marks] C Programming**

- (a) Write a function `int palindrome(char *str, int len)` which for the given input string `str` of length `len` determines if it is a palindrome or not. A palindrome is defined as “a sequence of characters (word, phrase, number) that has the property of being the same in either direction”.

Examples of palindromes	These are NOT palindromes
hannah	Hannah
tumut	Tumut
42level24	42leve142
racecar	RaceCar

Your function should return 1 if the input string `str` is a palindrome, and 0 if it is not. Your program must NOT contain any calls to standard C library functions.

[6 marks]

**Question 2 (continued)**

- (b) Below you can see two different implementations of a function `toupper` that are claimed to convert lowercase letters in the given input string `str` into uppercase letters.

```

void toupper(char str[]) {
    int i = 0;

    while (str[i] != 0) {
        if ((str[i] > ('a'-1)) &&
            (str[i] < ('z'+1)))
            str[i] ^= 0xDF;
        i++;
    }
}

char *toupper(char *str) {
    int i;

    do {
        if ((str[i] >= 'a') &&
            (str[i] <= 'z')) {
            str[i] = str[i] + 32;
        }
        i = i + 1;
    } while (str[i] != '\0');

    return str;
}

```

Which of the two functions is correct? What is wrong with the other function? Describe three errors.

[3 marks]

- (c) (i) Write a C statement that assigns the address of the double variable `sum` to the double pointer `tmp`.

(i)

[1 mark]

- (ii) Write a C statement that declares a pointer initialized to the text string "Hello world!" called `msg`.

(ii)

[1 mark]

**Question 2 (continued)**

- (d) Write a sequence of C statements that define a structure called `date` consisting of the variables `day` (of type integer), `month` (a string of length 10) and `year` (again an integer). Next, define a variable `mydates` as an array of 10 elements of `date`. Finally, assign the following two dates to the first two elements in the `mydates` array:

17, June, 2006  
28, February, 2000

[2 marks]

- (e) (i) Write a C statement that defines a file handler called `myfile` which is a pointer to type `FILE`.

(i)

[1 mark]

- (ii) Write a C statement that, using the file handler `myfile`, opens a text file called `"mytextfile.txt"` for writing.

(ii)

[1 mark]

**Question 3 [20 marks] Assembly Level Machine Organisation**

(a) Which of the following six assembly instructions is a valid PeANUt instruction? Explain in one sentence.

- |               |              |            |
|---------------|--------------|------------|
| A: loadpsw 42 | C: xor #42   | E: bov 42  |
| B: store #42  | D: setxr #42 | F: clov 42 |

[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 a200 (octal) onwards (containing values between 0 and 9, i.e. a200 contains 0, a201 contains 1, and so on).

```

loada    125    ; a175
add      #3
storexr
load     *3
add     #'0'
trap    #3
    
```

(i)

[1 mark]

(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]

(iii) What character is printed by the above PeANUt code?

(iii)

[1 mark]

**Question 3 (continued)**

- (c) (i) Write a sequence of PeANU<sub>t</sub> 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 = 0;

while (a < 10) {
    d = a + b + c;
    c = d;
    a = a + 1;
}
```

(i)

[5 marks]

- (ii) Write down the value of the integer `d` after the above program has been executed.

(ii)

[1 mark]

**Question 3 (continued)**

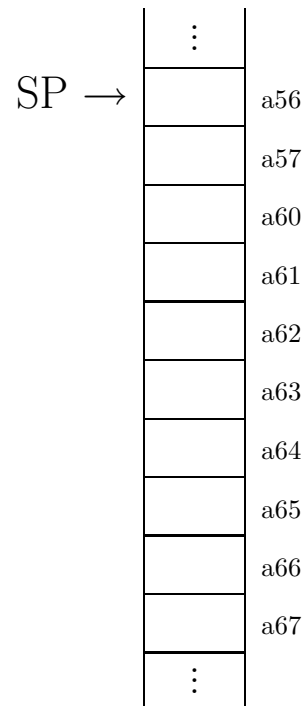
(d) Assume you have a PeANUt assembly program that consists of a main part and a function `myst` as shown below. Analyse the given assembly code and answer the questions (i) to (viii) on the following pages. On the right side you see parts of the PeANUt memory. The *stack pointer (SP)* is shown pointing to the memory cell with address `a56` at the beginning of the program (i.e. after the PeANUt has been initialised and the program has been loaded, but before the program has been started).

```

a:      block      4

        RV =      -2
        x =      -1
myst:   load       !x           ; <3>
        storexr
        load      *0
        store    !RV
        load      *1
        cmp      !RV
        ble      myst2
        store    !RV
myst2:  load       *2
        cmp      !RV
        ble      myst3
        store    !RV
myst3:  load       *3
        cmp      !RV
        ble      myst4
        store    !RV
myst4:  ret

main:   load      #0
        storexr
        add      #1
        store    *a
        mul      #4
        incxr   #1
        store    *a
        sub      #1
        incxr   #1
        store    *a
        mul      a+1
        sub      #3
        store    *a+1          ; <1>
        incsp   #1           ; <2>
        loada   a
        incsp   #1
        store    !0
        call    myst
        incsp   #-1
        load    !0
        ...     ...         ; <4>
        add     #'0'         ; <5>
        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) What type of variable is `a`?

(i)

[1 mark]

(ii) Give the value (or values) in `a` after the instruction at `<1>` (`store *a+1`) has been executed.

(ii)

[1 mark]

(iii) What is the purpose of the instruction `incsp #1` at `<2>`?

(iii)

[1 mark]

(iv) How does the stack look like at the beginning of the `myst` function at `<3>` (after the `call` instruction has been executed)? **Write your answer directly into the stack diagram on the previous page.**

[1 mark]

(v) What is the correct missing instruction at `<4>` so that the program follows the PeANUt procedure call convention?

(v)

[1 mark]

(vi) What is the purpose of the instruction `add #'0'` at `<5>`?

(vi)

[1 mark]

(vii) What result will be printed by the PeANUt program given on the previous page?

(vii)

[1 mark]

**Question 3 (continued)**

(viii) Write a **C** function that does the same as the `myst` PeANUt function on page 10.

(viii)



[2 marks]

(e) Assume the PeANUt memory at address `a10` contains the bit pattern **01010101 10101010**, and at address `a20` the decimal value **255** is stored. The following three instructions are executed:

```
load 16 ; a20
xor 8 ; a10
not
```

What binary value (16 bits) is stored in the accumulator after these three instructions have been executed? Clearly show your workings.



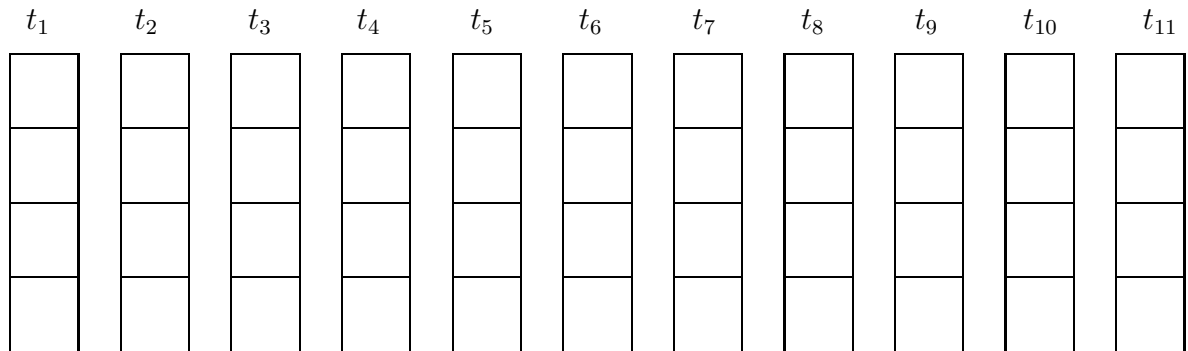
[1 mark]

**Question 4 [15 marks] Memory Systems and Modern Machines**

(a) Given a virtual memory system with a main memory that can hold four memory pages. Assume the *least recently used* page replacement policy is used by the memory management unit. The following sequence of 11 page accesses at times  $t_1$  to  $t_{11}$  to 6 different pages (numbered **1** to **6**) is given:

Time:	$t_1$	$t_2$	$t_3$	$t_4$	$t_5$	$t_6$	$t_7$	$t_8$	$t_9$	$t_{10}$	$t_{11}$
Page:	<b>6</b>	<b>3</b>	<b>2</b>	<b>1</b>	<b>6</b>	<b>4</b>	<b>1</b>	<b>5</b>	<b>1</b>	<b>3</b>	<b>2</b>

(i) Assuming the main memory has been empty at the beginning, complete the following diagram of the four main memory pages at times  $t_1$  to  $t_{11}$ . Please write your answers – the page numbers – into the appropriate boxes.



[2 marks]

(ii) How many pages have to be replaced in this page access sequence?

(ii)

[1 mark]

(b) Why do modern operating systems support virtual memory. Give two reasons (one sentence each).

[2 marks]

**Question 4 (continued)**

- (c) Briefly explain what tradeoffs there are in choosing an optimal page size in virtual memory. State two other contexts in computer systems design where a similar concept may be applied.

**[3 marks]**

- (d) Give two typical characteristics of Complex Instructions Set Computers and illustrate these with examples from the x86 (IA32) instruction set.

**[2 marks]**

## Question 4 (continued)

- (e) Consider the SPARC assembler program `example0.s` below, annotated with the corresponding C program in comments (the C code can assumed to be equivalent). The program is compiled using the commands

```
gcc -c example0.s; gcc -o example0.exe example0.o
```

```

        .align 4                !
        .type    f,#function    !
        .global f                !
f:      ! int f(int x) {
        save    %sp, -96, %sp    !
        sub     %i0, 1, %o0      ! // (x-1) into %o0
        sub     %i0, 7, %o1      ! // (x-7) into %o1
        call    .mul              ! // after call, (x-1)*(x-7) is in %o0
        nop                    !
        sub     %i0,11,%o1       ! // (x-11) into %o1
        call    .div              ! // after call, ((x-1)*(x-7))/(x-11) is in %o0
        nop                    !
        mov     %o0, %i0         ! // result in %o0; store in i0
        ret     !                ! return ((x-1)*(x-7))/(x-11);
        restore !                ! }
        .global main            !
        .type    main,#function !
        .align 4                !
main:   ! int main() {
        save    %sp, -96, %sp    ! static char *str = "f(9) = %d\n";
        mov     9, %l0           ! int v = 9; // in register %l0
        mov     %l0, %o0         !
        call    f                ! v = f(9);
        nop                    !
        mov     %o0, %l0         !
        sethi  %hi(str), %o1     ! printf(str, v);
        or     %o1,%lo(str), %o0 !
        mov     %l0, %o1         !
        call    printf           !
        nop                    !
        mov     1,%g1           ! exit(0);
        ta     0                ! }
        .section ".rodata"      !
str:    .asciz "f(9) = %d\n"    !

```

- (i) Write down the output produced by the command `./example0.exe`

(i)

[1 mark]

**Question 4 (continued)**

(ii) Rewrite the assembler code for the function `f()` so that the `nop` instructions (which are in *branch delay slots*) are eliminated (and the function returns the same result).

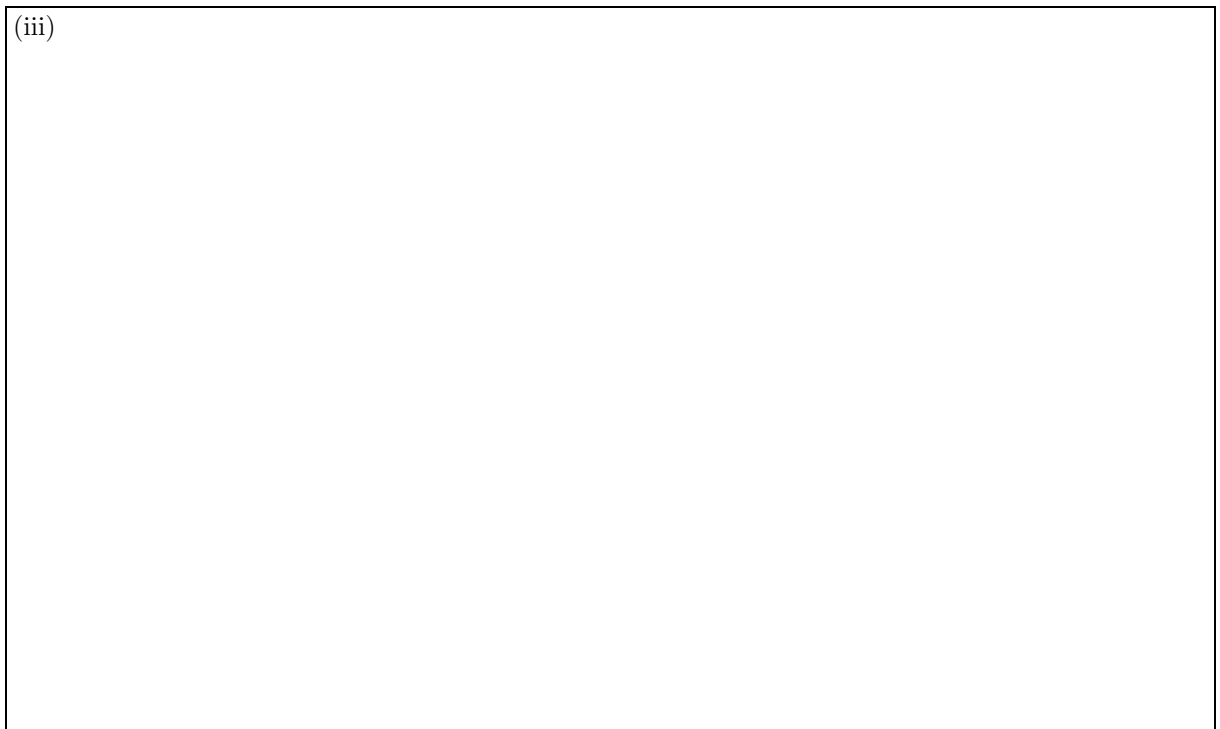
(ii)



[2 marks]

(iii) Consider the instruction sequences involved in function calls inside `example1.o`. In broad terms, how do these differ from those on the x86 architecture? Briefly explain why the support for function calls was an important feature in the design of the the SPARC instruction set.

(iii)



[2 marks]

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

(a) Consider the example SPARC assembler program of Question 4(e) on page 15.

(i) Write a table for the symbol names you would expect to find in `example0.o` (include only the symbols which explicitly appear in `example0.s`). Each table entry should contain the symbol's name, its scope (local, global or external global) and the ELF section it would belong to (`text`, `data`, `rodata`, `BSS` or 'undefined'). For those whose section is 'undefined', state which ELF section you would expect it to belong to in `example0.exe`.

(i)



[3 marks]

(ii) It is the intention that the function `f()` should not be accessed externally to `example0.s`. Describe how both the C and assembler code should be modified accordingly to reflect this. Briefly explain why it is better programming practice to do this.

(ii)



[2 marks]

**Question 5 (continued)**

- (b) Operating systems share CPU resources by *timeslicing*. What is a typical value (in seconds) for a timeslice on a multiuser computer? What might cause a process to utilize the CPU for less than a timeslice?

[2 marks]

- (c) IP addresses are often expressed in the decimal dot notation. For example, the host `www.anu.edu.au` has the IP address `150.203.99.8`. Write a mathematical expression (e.g. a sum of various powers of 2) which denotes the actual value this address represents. Briefly explain why IP addresses are structured in this way.

[2 marks]

**Question 5 (continued)**

- (d) Briefly explain one similarity between Virtual I/O and the layered computer communications architecture.

[1 mark]

Continuation of answer to Question 5 Part

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	$v$	000	—	illegal instruction
One	$v$	001	Load	$AC := OP$
One	$v'$	010	Store	$memory[AOP] := AC$
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	$f_1$	101000	Jump	jump to address AOP
Two	$f_1$	101001	BranchEqual	branch to AOP if EQ=1
Two	$f_1$	101010	BranchNotEqual	branch to AOP if EQ=0
Two	$f_1$	101011	BranchGreater	branch to AOP if GT=1
Two	$f_1$	101100	BranchLessEqual	branch to AOP if GT=0
Two	$f_1$	101101	BranchOverflow	branch to AOP if OV=0
Two	$f_1$	101110	LogicalAnd	$AC := AC \wedge OP$ , bitwise
Two	$f_1$	101111	LogicalOr	$AC := AC \vee OP$ , bitwise
Two	$f_1$	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	$f_1$	110100	CallProcedure	call procedure at OP
Two	$f_0$	110101	Trap	perform trap number OP
Two	$f_1$	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

<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

$x$	$2^x$
-5	0.03125
-4	0.0625
-3	0.125
-2	0.25
-1	0.5
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