THE AUSTRALIAN NATIONAL UNIVERSITY

First Semester Examination, June 2009

COMP2300 / COMP6300
(Introduction to Computer Systems)

Writing Period: 3 hours duration

Study Period: 15 minutes duration
Permitted Materials: One A4 page with notes on both sides.
NO calculator permitted.
Answer all of Questions 3–5; Questions 1 and 2 are optional.

For each of the optional Questions 1–2, the maximum of your mark for the question and the corresponding question of the Mid-Semester Exam will be used to determine your final assessment for this course.

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.

More marks are likely be awarded for answers that are short and concrete than for answers of a sketchy or rambling nature. Answers which are not sufficiently legible may not be marked.

Note that in some parts of the exam, there will be different questions asked of COMP2300 and COMP6300 students.

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

Student Number:

Official use only:

Q1 (13)	Q2 (17)	Q3 (25)	Q4 (20)	Q5 (15)	Total (90)

QUESTION 1 [13 marks]

(a) Assume memory addresses `0x02000411` to `0x02000415` contain the following 8-bit binary values:

Address	0x02000411	0x02000412	0x02000413	0x02000414	0x02000415
Binary value	0010 1100	1000 1101	0100 1000	0110 1011	0011 0000

Assume `sizeof(x) = 2` and `sizeof(y) = 2`, `&x = 0x02000412` and `&y = 0x02000414`, 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.

QUESTION 1(a)[i]	**[2 marks]**

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

```
printf("Value for (x+y)/256: %d", (x+y)/256);
```

Clearly show how you derive your answers.

QUESTION 1(a)[ii]	**[1 mark]**

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 (with an implicit leading bit). Convert the fractional decimal number -9.125 into IEEE single-precision floating point format. Give your answer in binary and hexadecimal.

QUESTION 1(b)	**[2 marks]**

(c) Give an example of integer overflow upon addition for each of signed and unsigned 4-bit integers. Include in your answer the bit patterns and decimal values of the operands and results.

QUESTION 1(c)	**[2 marks]**

Question 1 (continued)

(d) Name the six levels of the abstract machine (or computer architecture). Briefly explain in what sense this forms an abstraction.

QUESTION 1(d) **[4 marks]**

(e) Briefly explain why, in the instruction set design of RISC computers, the only instructions that can directly access memory are load and store instructions.

QUESTION 1(e) **[2 marks]**

QUESTION 2 [17 marks]

(a) Given the declaration `int t, x, y;` and these variables have all been initialized, explain the purpose of the following C code fragment:

```
t = x; x = y; y = t;
```

QUESTION 2(a)	**[1 mark]**

(b)
```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[]) {
   int i; unsigned int s=0;
   int n = atoi(argv[1]);
   for (i=0; i<n; i++)
     s += i+1;
   printf("%u\n", s);
   return 0;
}
```

Suppose the program was compiled and linked into an executable program called `foo`.

(i) Write the output produced by the command `./foo 4` (*hint:* in this case, the variable `n` will have the value of 4).

QUESTION 2(b)[i]	**[1 mark]**

(ii) Suppose n represents an integer. In terms of n, state what the command `./foo` n produces.

QUESTION 2(b)[ii]	**[1 mark]**

Question 2 (continued)

(c) Consider the following C function definition.

```
char *reverse(char *s, const char *t);
// the elements of the string t are copied into s in
// reverse order. It returns a pointer to s
```

For example, `reverse(s, "abc")` would set `s` to `"cba"`.

(i) Write an implementation of `reverse()`. Your code must not call any other function except `strlen()`.

QUESTION 2(c)[i]	**[3 marks]**

(ii) It is quite common in C string functions to modify a parameter *and* return a pointer to that (result) parameter. Briefly explain why it is useful to do so.

QUESTION 2(c)[ii]	**[1 mark]**

(iii) C provides an `assert()` facility. Illustrate how the facility could be used in this function.

QUESTION 2(c)[iii]	**[1 mark]**

Question 2 (continued)

(iv) What is one possible problem that can occur where functions like `reverse()` are used that cannot be detected by a facility such as `assert()`?

QUESTION 2(c)[iv]	**[1 mark]**

(v) Suppose we redefined `reverse()` as follows:

```
char *reverse(const char *t);
// returns a pointer to a new string which has the
// elements of the string t copied into it in reverse order
```

What modifications of the code from part (a) would be required to accomplish this? (for full marks, write the code changes). What problem might this redefinition introduce?

QUESTION 2(c)[v]	**[2 marks]**

(d) State two reasons why C is a good programming language for computer systems.

QUESTION 2(d)	**[2 marks]**

(e) Suppose a C program `shuffle2` had the usage:

```
shuffle2 [-p nrow] [-P k0] n k
```

with the following default values: `nrow = 0`, `k0 = 1`. Supposing the C source file for this program had the following declarations,

```
int main(int argc, char *argv[]) {
  int n, k, nrow, k0;
  ...
}
```

Write C code for `main()` that would initialize the variables `n, k, nrow` and `k0` from the command line parameters (stored in `argv[0]` .. `argv[argc-1]`) according to the above usage. Declare any new variables that you introduce. Your code need not check for any errors in the command line parameters. You code may use the standard `int atoi(char *)` function.

QUESTION 2(e)	**[4 marks]**

QUESTION 3 [25 marks]

(a) On the PeANUt machine, assume the accumulator **AC** contains a decimal value of `13`, the stack pointer register **SP** contains a decimal value of `4`, the index register **XR** contains a decimal value of `3` and the memory cells at addresses `0` to `3` contain the decimal values `42`, `0`, `7` and `66` respectively. Explain in detail what happens when the PeANUt machine executes the instruction `add *0`, which is at address decimal 70.

Your answer should describe the events that happen in the phases *Fetch*, *Evaluate Operand* and *Execute* and the values moved between the components **AC**, **SP**, **XR**, **CI**, **MAR**, **MDR** and **ALU**. All values should be expressed in binary or decimal.

QUESTION 3(a)	**[4 marks]**

(b) Give a few lines of PeANUt machine language code to read in the next character from standard input, add the value decimal 32 to it, and store the resulting character in memory location `a10`.

QUESTION 3(b)	**[2 marks]**

Question 3 (continued)

(c) Consider the PeANUt code fragment:

```
x:      data    14
y:      data     6
z:      block    1
        ...
        load     x
        dvd      y
        mul      y
        mul      #-1
        add      x
        store    z
```

(i) After the fragment is executed, what is the value in memory location z?

QUESTION 3(c)[i]	**[1 mark]**

(ii) How many memory references are required in the execution of the above code fragment?

QUESTION 3(c)[ii]	**[1 mark]**

(iii) In a context where x and y could contain arbitrary values, state in plain English the purpose of the instructions above.

QUESTION 3(c)[iii]	**[1 mark]**

Question 3 (continued)

(d) Consider the execution of the PeANUt program:

```
a:      data    24
        RV      = -3
        x       = -2
        y       = -1
m:      data    15
myst:   load    !x
        add     !y
        and     m
myst1:  store   !RV
        ret
main:   incsp   #3
        load    a
        store   !-1
        load    #17
        store   !0
        call    myst
        load    !-2
        incsp   #-3
        trap    #1
end     main
```

(i) Draw the PeANUt stack frame at the point where the first instruction of `myst` is executed. Clearly indicate which cell the stack pointer points to, and include symbolic labels and values to all relevant cells.

QUESTION 3(d)[i]	**[3 marks]**

Question 3 (continued)

(ii) What is the value of the accumulator **AC** when the `trap #1` instruction is executed?

QUESTION 3(d)[ii]	**[2 marks]**

(iii) Write a C code implementation for the function `myst`. All variables may be assumed to be of type `int`.

QUESTION 3(d)[iii]	**[1 mark]**

(e) The following part asks you to translate C code fragments into PeANUt assembly language. In all cases, there is no need to translate any implied C declarations into assembler; assume that the `block` directive has been used to do this. Define any macros that you use. There is no need to comment your assembly code. Divide the answer boxes below into two columns to make sufficient room.

(i) Translate the C code fragment of Question 2(a) into PeANUt assembly language.

QUESTION 3(e)[i]	**[1 mark]**

Question 3 (continued)

(ii) Translate the `for` loop of Question 2(b) into PeANUt assembly language.

QUESTION 3(e)[ii] **[3 marks]**

(iii) Translate the following C function into PeANUt assembly language.

```
void printElt(int i, int a[]) {
  if (a[i] >= 0)
    printf("%u\n", a[i]);
}
```

Assume that the parameters are passed using the standard PeANUt procedure call convention. Assume there is an external function `WriteCard` available that follows the standard PeANUt procedure call convention, where `WriteCard(x)` has the same effect as `printf("%u",x)`.

QUESTION 3(e)[iii] **[4 marks]**

(f) COMP2300 students, only, answer this question:

Briefly explain why procedure call conventions, as for example in the PeANUt, are needed.

COMP6300 students, only, answer this question:

The PeANUt procedure call convention uses the stack to pass the return value. Name an alternative convention for passing the return value that could be used on the PeANUt, and discuss its relative merits.

QUESTION 3(f)	**[2 marks]**

Additional answers to QUESTION 3 (__)[__]

QUESTION 4 [20 marks]

(a) Consider a virtual memory system with a main memory that can hold four memory pages. Assume that the *First-In-First-Out* (*FIFO*) page replacement policy is used by the operating system. 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:	**1**	**2**	**1**	**4**	**1**	**5**	**2**	**6**	**1**	**2**	**1**

(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.

t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}

[3 marks]

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

QUESTION 4(a)[ii] **[1 mark]**

(iii) If the *least recently used* policy was used instead, what would the number of page replacements? State the reason why in general this policy performs better than FIFO.

QUESTION 4(a)[iii] **[2 marks]**

Question 4 (continued)

(iv) Describe one situation where the random replacement policy will perform better (i.e. produce fewer page faults) than the least recently used policy.

QUESTION 4(a)[iv]	**[2 marks]**

(b) Page table entries typically contain a *dirty bit*. Briefly describe its purpose. Is a similar piece of information required for cache memory? Briefly explain.

QUESTION 4(b)	**[2 marks]**

Question 4 (continued)

(c) Briefly describe how memory protection can be implemented in conjunction with *paging*, both within a process, and between two processes that are being executed at the same time.

QUESTION 4(c) **[2 marks]**

(d) What is a branch delay slot, and why were they introduced to RISC processors?

QUESTION 4(d) **[2 marks]**

Question 4 (continued)

(e) Explain why the *latency* of a disk read or write, i.e. the time to access the first byte of a sector, can be relatively large. Explain how it is possible to achieve a reasonably high rate of data transfer for subsequent bytes.

COMP6300 students, only, answer the following question as well.

Briefly explain what implications this has for file organization.

QUESTION 4(e) **[3 marks]**

(f) Consider the following function.

```
int sum(int n, int a[]) {
  int s = 0, i;
  for (i = 0; i < N ; i++)
    s += a[i];
  return s;
}
```

Briefly describe what differences you would expect in terms of memory accesses if the above code was executed on the SPARC and x86 architectures.

QUESTION 4(f) **[3 marks]**

Additional answers to QUESTION 4 (__)[__]

QUESTION 5 [15 marks]

(a) List two important advantages that multiprocessing provides.

QUESTION 5(a) **[2 marks]**

(b) Briefly describe two situations where *interrupts* provide crucial support to the operating systems.

QUESTION 5(b) **[2 marks]**

Question 5 (continued)

(c) COMP2300 students, only, answer the following question:

Consider the following C programs:

```c
#include <string.h>
int main(void) {
  static char month[9];
  strcpy("December", month);
  //dest. str. should go 1st!
  return 0;
}
```

```c
#include <stdio.h>
int main(void) {
  char month[9];
  printf("%s\n", month);
  return 0;
}
```

When compiled and run, the program on the left crashed with a segmentation violation (i). When compiled and run, the program on the right printed out a seemingly random sequence of characters (ii). However, if the program was modified so that the variable declaration of `month` was the same as for the program on the left, an empty line was printed out (iii). Briefly explain the above three behaviors.

COMP6300 students, only, answer the following question:

In C, local variables are allocated by default on the stack. Describe the advantages and disadvantages (including potential for programmer error) of this.

QUESTION 5(c)	**[4 marks]**

Question 5 (continued)

(d) When a compiled C program is executed, the `main()` function is called. Briefly describe what happens during execution before and after this, and why this is needed.

QUESTION 5(d) **[2 marks]**

(e) Briefly explain the relation between a *system call* and a *trap*.

QUESTION 5(e) **[1 mark]**

Question 5 (continued)

(f) *For this part, answer only one of the following questions. Either:*

Define the concept of *virtualization* in computer systems. Briefly describe three distinct examples of virtualization.

Or:

Describe what common concepts and issues are shared between packets (in the context of networking), disk blocks and cache lines.

QUESTION 5(f)	**[4 marks]**

Additional answers to QUESTION __(__)[__]

Additional answers to QUESTION __(__)[__]

Additional answers to QUESTION __(__)[__]

Additional answers to QUESTION __(__)[__]

Additional answers to QUESTION ___(___)[___]

Additional answers to QUESTION ___(___)[___]

Appendix

(this page may be detached for your convenience)

format	*mode*	*opcode*	*name*	*meaning*
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

Modes: Immediate is 000, Direct is 001, Indirect is 010, Indexed is 011, Stack is 100.

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*
`Load`	`load`
`Store`	`store`
`Addition`	`add`
`Subtraction`	`sub`
`Division`	`dvd`
`Multiplication`	`mul`
`Compare`	`cmp`
`Jump`	`jmp`
`BranchEqual`	`beq`
`BranchNotEqual`	`bne`
`BranchGreater`	`bgt`
`BranchLessEqual`	`ble`
`BranchOverflow`	`bov`
`LogicalAnd`	`and`
`LogicalOr`	`or`
`LogicalXor`	`xor`

machine instruction	*operation*
`SetIndexReg`	`setxr`
`IncIndexReg`	`incxr`
`IncStackPoint`	`incsp`
`CallProcedure`	`call`
`Trap`	`trap`
`LoadAddress`	`loada`
`Return`	`ret`
`ClearOver`	`clov`
`LoadPSW`	`ldpsw`
`StorePSW`	`stpsw`
`LogicalNot`	`not`
`CompareIndexReg`	`cmpxr`
`LoadIndexReg`	`loadxr`
`StoreIndexReg`	`storexr`
`LoadStackPoint`	`loadsp`
`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

x	2^x
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