THE AUSTRALIAN NATIONAL UNIVERSITY

First Semester Examination, June 2008

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.

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:

Official use only:

Q1 (12)	Q2 (18)	Q3 (25)	Q4 (20)	Q5 (15)	Total (90)

QUESTION 1 [12 marks]

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

Address	0x02000411	0x02000412	0x02000413	0x02000414	0x02000415
Binary value	0010 0110	1000 1101	1100 1000	0110 0111	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: %d", x+y);
```

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 -13.625 into IEEE single-precision floating point format. Give your answer in binary and hexadecimal.

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

(c) Give an integer expression for the approximate value of the largest number expressible in IEEE single-precision floating-point. Suppose it was desired to represent numbers up to 16 times larger than this; how would you change the format to accommodate this? What tradeoff would be involved?

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

Question 1 (continued)

(d) Name and briefly describe the three main features of the Central Processing Unit (CPU) of a computer.

QUESTION 1(d) **[3 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 [18 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 += 2*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]**

(c) Consider the following code fragment:

```
int x[] = {1, 2, 3, 3, 3};
int i, b[5], s = 0;
for (i=0; i < 5; i++)
  b[i] = 0;
for (i=0; i < 5; i++)
  b[x[i]] = 1;
for (i=0; i < 5; i++)
  if (b[i]) s++;
```

After this code executes, what is the value of `s`?

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

(d) Suppose the code of part (c) above was generalized so that `x[]` and `b[]` were arrays of length $n > 0$ and the elements of `x[]` were initialized to have any value in the range 0 to $n-1$. Describe in plain English what the final value of `s` is in terms of the elements of `x[]`.

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

(e) Consider the following standard C function definition.

```
char * strcpy(char *dest, const char *src);
// copies the string pointed to by src (including the terminating
// '\0' char) to the string pointed to by dest.
// It returns a pointer to the destination string dest
```

Write an implementation of `strcpy()`. Your code must not call any other function.

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

(f) If any two elements, not necessarily consecutive, in an array of integers are out of order (i.e. the element occurring first in the array is greater than the element occurring second), that is called an *inversion*. For example, given the declaration

```
int x[] =  {4, 5, 6, 2, 1, 3};
```

then the array `x[]` has 10 inversions as:

```
x[0]=4  >  x[3]=2, x[4]=1, x[5]=3
x[1]=5  >  x[3]=2, x[4]=1, x[5]=3
x[2]=6  >  x[3]=2, x[4]=1, x[5]=3
x[3]=2  >  x[4]=1
```

The skeleton code below is intended to count the number of inversions in `x[]`, which is of length `n`.

```
int i, j, inversionCount = 0;
for (i=0; i < n; i++)
   for ( XXXXXXX )
     if (x[i] > x[j])
       inversionCount++;
```

Write the code that should be in the place of `XXXXXXX`.

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

(g) Consider the following function definition:

```
// assumes s[] contains a list of positive integers in ascending
// order followed by a 0. returns the lowest value i >= 0 such
// that s[i] >= e  (or, if e is larger than than all elements,
// it returns the position of the 0)
// example: if s[] = {2, 4, 6, 8, 0},
//   firstPos(6, s) = 2, firstPos(7, s) = 3, firstPos(9, s) = 4
int firstPos(int e, int *s) {
  int i=0;
  while ( XXXXX )
    i++;
  return i;
}
```

Write the correct Boolean condition for the `while` loop in `firstPos()`.

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

(h) Suppose an integer array `c` of length `C` is organized so that an integer item `i>=0` may only be stored in the elements `c[itag*K]`, `c[itag*K+1]`, ..., `c[itag*K+K-1]`, where `itag = i % (C/K)`. It may be assumed that $K > 0$ and $C \% K = 0$. Write a function whose header is given by:

```
int isIn(int i, int c[], int C, int K);
```

that returns 1 if `i` is in `c[]`, and 0 otherwise. Your code should not access any elements of `c[]` that cannot contain the value of `i`.

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

Question 2 (continued)

(i) Suppose it is desired that the function `isIn()` is to be implemented in a file `isin.c`, and is to be called from the C program `testisin.c`. Both of these files are to be compiled separately. Briefly describe how this small system of programs should be arranged so that the C compiler can check whether the call of `isIn()` is consistent with its implementation (*hint:* make use of a header file).

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

(j) C provides an `assert()` facility. Explain why in general it is useful to provide such a facility. Illustrate how the facility could be used in the `strcpy()` function of Question 2(e).

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

QUESTION 3 [25 marks]

(a) On the PeANUt machine, assume the accumulator **AC** contains a decimal value of `10`, the stack pointer register **SP** contains a decimal value of `4` 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 !-2`, 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 affected components **AC**, **SP 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 and store this in memory location `a10`.

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

Question 3 (continued)

(c) Consider the PeANUt code fragment:

```
x:      data    14
y:      data     9
z:      block   1
        ...
        load    x
        sub     y
        store   z
```

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

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

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

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

(d) Consider the PeANUt code fragment:

```
One:    data    1
str:    data    "012345"    ; note: ascii value of '0' = 48
        block   1
v:      block   1
        ...
        setxr   #0
lab1:   load    *str
        cmp     #0
        beq     lab2
        load    *str
        and     One
        add     v
        store   v
        incxr   #1
        jmp     lab1
lab2:
```


Question 3 (continued)

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

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

(ii) Translate the executable code of the fragment into an equivalent C code fragment (your C code may assume the declarations `char *str;int v;`)

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

(iii) Explain in plain English what your C code fragment computes.

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

(e) Consider the execution of the PeANUt program:

```
a:      data    11
b:      data     7
        RV      = -3
        x       = -2
        y       = -1
myst:   load    !x
        cmp     !y
        ble     myst1
        load    !y
myst1:  store   !RV
        ret
main:   incsp   #3
        load    a
        store   !-1
        load    #13
        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(e)[i]	**[3 marks]**

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

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

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

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

(f) 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(f)[i]	**[1 mark]**

Question 3 (continued)

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

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

(iii) Translate the `printf("%u\n", v)` call of Question 2(b) into PeANUt assembly language. 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(f)[iii] **[1 mark]**

Question 3 (continued)

(iv) Translate the assigment statement `b[x[i]] = 1;` of Question 2(c) into PeANUt assembly language.

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

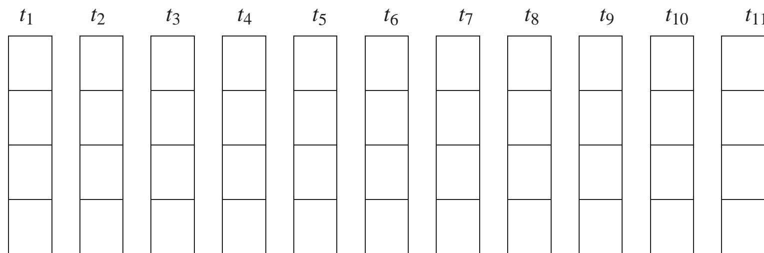
Additional answers to QUESTION 3 (__)[__]

QUESTION 4 [20 marks]

(a) Consider a virtual memory system with a main memory that can hold four memory pages. Assume the *least recently used* 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**	**3**	**4**	**5**	**6**	**1**	**5**	**6**	**3**	**2**

(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.



[3 marks]

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

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

Question 4 (continued)

(b) Consider a memory access pattern which repeatedly accesses pages **1**, **2**, ..., P with a main memory consisting of 4 pages. Assuming the main memory is empty at the beginning, give the eventual percentage of page misses per access for the LRU policy when (i) $P = 6$, and (ii) $P = 8$. Briefly explain why this is the case.

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

(c) Consider the situation of part (b) when a random replacement policy is used. Estimate (as best you can) the eventual percentage of page misses per access when (i) $P = 4$, and (ii) $P = 8$. Briefly explain why the random policy begins to lose its advantage as P increases.

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

Question 4 (continued)

(d) In virtual memory, pages tables are normally kept in reserved areas of physical memory. This introduces a potential problem: every instruction which accesses virtual memory will require at least two physical memory accesses: one to access the page table, and one to access the corresponding page in physical memory.

(i) What recent trend in CPU and memory technology makes this particularly problematic?

QUESTION 4(d)[i] **[1 mark]**

(ii) Briefly describe how is this problem typically solved on a modern computer. Name another typical feature of modern computers which is based on a similar principle.

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

(e) Consider the following argument. *Modern computers need most memory accesses to hit the caches in order to attain good performance. Therefore, if we wish for an executing program to have good performance, we must ensure that the caches are large enough to hold all of the code and data that might ever be required for that program simultaneously*. Briefly explain what is the flaw in the conclusion of this argument.

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

Question 4 (continued)

(f) List three examples of where *parallelism* is used in modern computer design. For each, state the degree of impact it (potentially) has on how high-level language software that is required to run efficiently on the computer needs to be written.

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

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

In the x86 instruction set, there are only 4 registers that may normally be used to hold integer and address data (`%eax`, `%ebx`, `%ecx`, `%edx`). Briefly describe the potential impact this has on the performance of programs when run on this architecture. Name one feature of the x86 instruction set that can compensate for this effect.

or:

In the context of the execution cycle of a RISC processor, briefly describe the problem conditional branch instructions present. Name the solution employed on modern RISC processors.

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

QUESTION 5 [15 marks]

(a) Consider the following C program `example0.c`. The program is compiled into an object file `example0.o` and finally linked into an executable called `example0`.

```
#include <stdlib.h>
int a[1024];
static int b[1024];
int main() {
  int *c = NULL;
  static int d[1024];
  int e[1024];
  c = malloc(1024 * sizeof(int));
  return 0;
}
```

For each identifier mentioned in the program, write a table, where each table entry should contain whether the identifier has a symbol table entry in `example0.o`, and if so its type ('function', 'object' or 'notype'), its symbol binding (local or global) and the ELF section (`.text`, `.data`, `.bss` or none) it occupies in `example0.o`. For the entries that are integer arrays, state which memory area in `example0` ('text', 'data', 'heap' or 'stack') you would expect the 0th element of the array to reside in (at the point just before the `return` statement is executed).

QUESTION 5(a) **[6 marks]**

Question 5 (continued)

(b) Consider the following code fragment:

```
int IsDirty(int pte) {
  char result[1];
  sprintf(result, "%d" , GetBits(pte, 4, 1) >> 4);
  ...
}
```

When compiled and run on the x86 architecture, an application using this function generated a segmentation violation shortly after the above function was first called. It can be assumed that the expression `GetBits(pte, 4, 1) >> 4` always has a value of 0 or 1. Note that the standard C `sprintf()` function is like `printf()`, except that it takes an extra first string parameter, and instead of the characters being printed to the standard output, they are printed into the string.

(i) State what the programming error is in the above code, and how you would fix it.

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

(ii) Given that the C compiler allocated `result` to the address `%ebp - 1`, briefly explain how the error caused the segmentation violation. *Hint:* consider the x86 procedure calling convention and stack frame layout.

QUESTION 5(b)[ii] **[1 mark]**

Question 5 (continued)

(iii) The above application with the above code ran without causing any apparent problems for many years on a SPARC processor. Note that the C compiler allocated `result` to the corresponding address on the stack (denoted `%fp - 1` in SPARC). Give a brief explanation for why this was the case.

QUESTION 5(b)[iii]	**[1 mark]**

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

State and briefly explain the two main advantages of *multiprocessing*.

or:

State and briefly explain he two main advantages of *virtual I/O*.

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

Question 5 (continued)

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

Briefly describe the three main 'abstraction layers' in the computer network communications architecture.

or:

Give a definition of the concept of *virtualization* as it applies to computer systems. Briefly describe two examples of virtualization.

QUESTION 5(d)	**[3 marks]**

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 := ¬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