

THE AUSTRALIAN NATIONAL UNIVERSITY

First Semester Examination, June 2007

COMP2300
(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:

Table with 6 columns: 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:

Table with 6 columns: Address, 0x02000411, 0x02000412, 0x02000413, 0x02000414, 0x02000415; Binary value

Assume sizeof(x) = 2 and sizeof(y) = 2, &x = 0x02000412 and &y = 0x02000414, and the data storage is big 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("Values for x+y: %d %u", x+y, x+y);

Clearly show how you derive your answers.

QUESTION 1(a)(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 (with an implicit leading bit). What floating point number is represented by the following 32-bit number (given in hexadecimal representation):

0xC2BA0000

For full marks, your answer should be expressed as a single decimal number accurate to four decimal digits.

|               |           |
|---------------|-----------|
| QUESTION 1(b) | [2 marks] |
|---------------|-----------|

- (c) The largest number expressible in IEEE single-precision floating-point is slightly less than  $2 \times 2^{128}$ . 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) | [1 mark] |
|---------------|----------|

**Question 1 (continued)**

- (d) Signed integers may be represented on a computer by reserving the topmost bit to represent the sign. Give two advantages that the two's complement representation has over this scheme.

|               |          |
|---------------|----------|
| QUESTION 1(d) | [1 mark] |
|---------------|----------|

- (e) Name and briefly describe the five main components of a typical computer (i.e. typical computer architecture).

|               |           |
|---------------|-----------|
| QUESTION 1(e) | [5 marks] |
|---------------|-----------|

**QUESTION 2 [17 marks]**

- (a) Write C statements to declare an integer variable `i` initialized to 0, and a pointer to an integer `pi` initialized to the address of `i`.

|               |          |
|---------------|----------|
| QUESTION 2(a) | [1 mark] |
|---------------|----------|

- (b) Write the output of this program, clearly indicating spaces.

```
#include <stdio.h>
#define N 8
int main(){
    int a[N];
    int i;
    a[0] = 0;
    for (i=1; i<N; i++) {
        a[i] = a[i-1] + i;
        printf("%3d", a[i]);
    }
    printf("\n");
    return 0;
}
```

|               |          |
|---------------|----------|
| QUESTION 2(b) | [1 mark] |
|---------------|----------|

**Question 2 (continued)**

- (c) Consider the following function definition.

```
int strlen(const char *s);
// returns the length of the string s,
// not including the terminating '\0' character
```

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

|               |           |
|---------------|-----------|
| QUESTION 2(c) | [2 marks] |
|---------------|-----------|

- (d) Write an implementation of the function `void printbin(unsigned int i)`, which prints `i` in binary form, with leading zeroes omitted. For example, `printbin(5)` should print exactly the three character sequence '101'.

|               |           |
|---------------|-----------|
| QUESTION 2(d) | [4 marks] |
|---------------|-----------|

**Question 2 (continued)**

(e) Suppose it is desired to create a C library file `printbin.c` which contains *only* the function `printbin()` of Question 2(d). As always, it is important that library files are compiled consistently with any files that use them.

(i) Write the corresponding header file `printbin.h`

|  |
|--|
| QUESTION 2(e)[i] <span style="float: right;">[1 mark]</span> |
|  |

(ii) Write the corresponding library file `printbin.c`. The code for the body of `printbin()` may be (should be) omitted (i.e. indicated by `'...'`).

|   |
|---|
| QUESTION 2(e)[ii] <span style="float: right;">[1 mark]</span> |
|   |

(iii) Write a simple C test program `testpb.c` which calls `printbin()` (it need do nothing else).

|  |
|--|
| QUESTION 2(e)[iii] <span style="float: right;">[1 mark]</span> |
|  |

(iv) Write a series of commands that would compile the system (on Linux) to create an executable program `testpb`.

|   |
|---|
| QUESTION 2(e)[iv] <span style="float: right;">[1 mark]</span> |
|   |

**Question 2 (continued)**

(f) Briefly describe the differences (aside from the different names) between the following two ways of providing code to determine the maximum of two numbers:

```
#define max1(a,b) ((a) > (b)? (a): (b))~
int max2(int a, int b) { return (a > b? a: b); }
```

|   |
|---|
| QUESTION 2(f) <span style="float: right;">[1 mark]</span> |
|   |

(g) 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 `strlen()` function of Question 2(c).

|  |
|--|
| QUESTION 2(g) <span style="float: right;">[2 marks]</span> |
|  |

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

|  |
|--|
| QUESTION 2(h) <span style="float: right;">[2 marks]</span> |
|  |

**QUESTION 3 [25 marks]**

- (a) On the PeANUt machine, assume the accumulator **AC** contains a decimal value of 10, 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 50.

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**, **XR**, **CI**, **MAR**, **MDR** and **ALU**.

QUESTION 3(a) [4 marks]

- (b) Why are *traps* used in PeANUt for input and output? Explain in one or two sentences. Give a few lines of PeANUt machine language code to print the newline (`'\n'`) character (which has an ASCII value of decimal 10).

QUESTION 3(b) [3 marks]

**Question 3 (continued)**

- (c) (i) Explain in two to three sentences what the following PeANUt assembly language code segment is doing. Assume that memory cells 10 to 19 contain the values 0, 1, ..., 9 respectively and the stack pointer **SP** = 128.

```
trap    #2
sub     #'0'
store  !0
add     13
```

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

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

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

- (iii) Assuming the character `'8'` was entered at the keyboard as the above code was executed, what will be the value of **AC** after the code segment is executed?

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

**Question 3 (continued)**

- (d) Translate the C program of Question 2(b) into an equivalent PeANUt assembly language program. Assume there is an external function `WriteInt` available in PeANUt that follows the standard PeANUt procedure call convention, where `WriteInt(x, 3)` has the same effect as `printf("%3d", x)`. Declare any macros that you use. There is no need to comment your assembly code. Divide the answer box below into two columns to make sufficient room.

|               |                   |
|---------------|-------------------|
| QUESTION 3(d) | <b>[10 marks]</b> |
|---------------|-------------------|

**Question 3 (continued)**

- (e) Translate the following PeANUt assembly language function into an equivalent C function. Note that the C code should reflect the fact that 16-bit integers are being manipulated. For full marks, the C code should use a bit shifting operation rather than a divide, and it should contain a comment stating in English what the function performs in terms of manipulating a bit pattern.

```
Mask:  data    255
      x      = -1

bitwise:
      load    !x
      storexr
      load    *0
      dvd    #16
      and    Mask
      store  *0
      ret
```

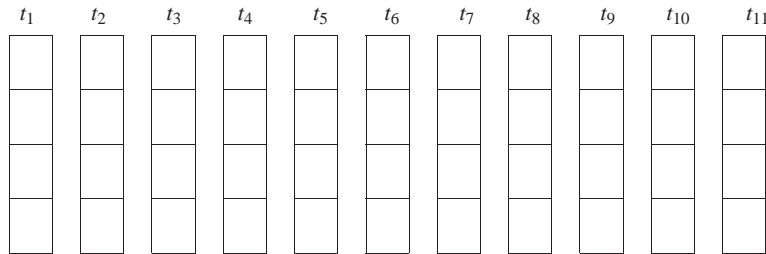
|               |                  |
|---------------|------------------|
| QUESTION 3(e) | <b>[4 marks]</b> |
|---------------|------------------|

**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: | 4     | 3     | 2     | 1     | 6     | 4     | 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) <span style="float: right;">[1 mark]</span> |
|   |

**Question 4 (continued)**

(b) In both paging and set-associative caches, the *least recently used* (LRU), *first-in-first-out* (FIFO) and random replacement policies may be used. In general, which of these performs best and which of these performs worst? What general property of the memory access patterns of programs gives rise to this? Briefly explain why. If one policy is generally better, explain in the context of caches why it is not always used in modern computers.

|  |
|--|
| QUESTION 4(b) <span style="float: right;">[3 marks]</span> |
|  |

(c) Suppose you had a computer with a 512 KB level-2 cache. If a program had a *working set* of 1 MB, would it be possible that the program would run with a low rate of level-2 cache misses? Briefly explain your answer.

|  |
|--|
| QUESTION 4(c) <span style="float: right;">[2 marks]</span> |
|  |

(d) One characteristic of RISC computers is that they have fixed length instructions. Give one advantage and one disadvantage of using fixed length instructions.

|   |
|---|
| QUESTION 4(d) <span style="float: right;">[1 mark]</span> |
|   |

**Question 4 (continued)**

- (e) What is the difference between SRAM and DRAM memory? Name one advantage and one disadvantage of SRAM over DRAM. Where are SRAM and DRAM used in a typical modern computer?

QUESTION 4(e) [3 marks]

- (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]

**Question 4 (continued)**

- (g) Explain why cache sizes are always in powers of two.

QUESTION 4(g) [1 mark]

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

Give three concrete examples from the x86 instruction set that illustrate a typical feature of CISC computers. Name each feature.

or:

Give three concrete examples from the SPARC instruction set that illustrate a typical feature of RISC computers. Name each feature.

QUESTION 4(h) [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.exe`.

```
#include <stdlib.h>
static int a[1024];
int b[1024];
int *c = NULL;
int main() {
    static int e[1024];
    int f[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.exe` ('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) | <b>[6 marks]</b> |
|---------------|------------------|

**Question 5 (continued)**

- (b) Consider the statement *One of the functions of the operating system is to define the user interface (command line, GUI) of the computer*. Discuss to what extent this is true, and to what extent it is false.

|               |                  |
|---------------|------------------|
| QUESTION 5(b) | <b>[2 marks]</b> |
|---------------|------------------|

- (c) What is a *system call*? Give two examples of system calls. State the relationship between a system call and a trap.

|               |                  |
|---------------|------------------|
| QUESTION 5(c) | <b>[3 marks]</b> |
|---------------|------------------|

**Question 5 (continued)**

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

Describe the sequence of actions taken by an operating system kernel when it switches between processes.

or:

Give two advantages of DMA (Direct Memory Access) I/O over programmed I/O.

|               |           |
|---------------|-----------|
| QUESTION 5(d) | [2 marks] |
|---------------|-----------|

(e) In the TCP/IP communication protocol, data is transmitted in packets. Give two advantages that arise from breaking long messages into packets.

|               |           |
|---------------|-----------|
| QUESTION 5(e) | [2 marks] |
|---------------|-----------|

|  |
|--|
| Additional answers to QUESTION —(—)[—] |
|--|

|  |
|--|
| Additional answers to QUESTION —(—)[—] |
|--|

Student Number: .....

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)

| <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>f<sub>1</sub></i> | 101000        | Jump            | jump to address AOP           |
| Two           | <i>f<sub>1</sub></i> | 101001        | BranchEqual     | branch to AOP if EQ=1         |
| Two           | <i>f<sub>1</sub></i> | 101010        | BranchNotEqual  | branch to AOP if EQ=0         |
| Two           | <i>f<sub>1</sub></i> | 101011        | BranchGreater   | branch to AOP if GT=1         |
| Two           | <i>f<sub>1</sub></i> | 101100        | BranchLessEqual | branch to AOP if GT=0         |
| Two           | <i>f<sub>1</sub></i> | 101101        | BranchOverflow  | branch to AOP if OV=0         |
| Two           | <i>f<sub>1</sub></i> | 101110        | LogicalAnd      | AC := AC ∧ OP, bitwise        |
| Two           | <i>f<sub>1</sub></i> | 101111        | LogicalOr       | AC := AC ∨ OP, bitwise        |
| Two           | <i>f<sub>1</sub></i> | 110000        | LogicalXor      | AC := AC ⊕ OP, bitwise        |
| Two           | <i>f<sub>0</sub></i> | 110001        | SetIndexReg     | XR := OP                      |
| Two           | <i>f<sub>0</sub></i> | 110010        | IncIndexReg     | XR := XR + OP                 |
| Two           | <i>f<sub>0</sub></i> | 110011        | IncStackPoint   | SP := SP + OP                 |
| Two           | <i>f<sub>1</sub></i> | 110100        | CallProcedure   | call procedure at OP          |
| Two           | <i>f<sub>0</sub></i> | 110101        | Trap            | perform trap number OP        |
| Two           | <i>f<sub>1</sub></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                              |
|----------------------|-----------------------|---------------------------------------|
| <i>v</i>             | any mode (0-4)        | must be explicitly specified (3 bits) |
| <i>v'</i>            | any mode except 0     | must be explicitly specified (3 bits) |
| <i>f<sub>0</sub></i> | mode 0 (fixed)        | implicitly specified by opcode        |
| <i>f<sub>1</sub></i> | 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

| <i>x</i> | $2^x$   | <i>x</i> | $2^x$ |
|----------|---------|----------|-------|
| -5       | 0.03125 | 5        | 32    |
| -4       | 0.0625  | 6        | 64    |
| -3       | 0.125   | 7        | 128   |
| -2       | 0.25    | 8        | 256   |
| -1       | 0.5     | 9        | 512   |
| 0        | 1       | 10       | 1024  |
| 1        | 2       | 11       | 2048  |
| 2        | 4       | 12       | 4096  |
| 3        | 8       | 13       | 8192  |
| 4        | 16      | 14       | 16384 |
|          |         | 15       | 32768 |

Table 4: Powers of 2 in decimal