

THE AUSTRALIAN NATIONAL UNIVERSITY

First Semester Examination – June 2000

COMP2300

Introduction to Computer Systems

ENGN2213

Computer Organisation

Study Period: 15 minutes

Time Allowed: 3 hours

Permitted Materials: One A4 page with notes on both sides.

COMP2300: Answer ALL questions.

ENGN2213: Answer questions 1, 2, and 3 only.

Questions are NOT equally weighted.

The Appendix contains information about the PeANUt instruction set and a table of the powers of 2.

Question 1 [12 marks] Fundamental Concepts

- (a) What value is represented by the bit pattern 1110 1111 1001 1000, if it is interpreted as a 16-bit two's complement integer? Give your answer in decimal, showing your working. [2 marks]
- (b) Convert the decimal value 453_{10} to *hexadecimal*. Show your working. [2 marks]
- (c) Define the function of each of the following components of a CPU:
- Control Unit
 - ALU
- [4 marks]
- (d) List the steps in the *fetch-decode-execute* cycle. [4 marks]

Question 2 [12 marks] The C Programming Language

- (a) What is the output of the following C program. Explain your answer. [4 marks]

```
#include<stdio.h>

int qwerty(int a, int b, int z);

int main()
{
    int x = 0, y = 0, z = 0;

    x = 1;
    y = 2;

    z = qwerty(x,y,z);

    printf('z=%d\n',z);

    if (z!=3) {
        printf('ZAP!\n');
    } else {
        printf('POW!\n');
    }

    return -x;
}

int qwerty(int a, int b, int z)
{
    z = a + b;
    return a - b;
}
```

- (b) Write an implementation for the C function described by the following function prototype. [8 marks]

```
void obscure(char *s);
/* Pre: s is a null terminated string, of zero or more characters.
   Post: lower case alphabetic characters in s have been replaced
         by '-', and upper case alphabetic characters have been
         replaced by '+'

   Example: if z[]=''+-AB3yp4'' then after obscure(z) has been
            executed, z[] will be ''+--+3--4''.
```

```
*/
```

Question 3 [26 marks] PeANUt

- (a) Describe the bitwise AND operation. How would you use this operation to set each of the two leftmost bits of an 16-bit number to 0, while leaving the other bits unchanged? [4 marks]
- (b) Write a sequence of PeANUt assembly language instructions that has the same effect as the following piece of C code. [6 marks]

```
int x, y, z;
...
x = z = 0;
y = 512;
while (x < y) {
    z = z + x;
    x = x + 1;
}
```

- (c) Assume that the address of `x` is 150_8 , that location 150_8 contains the value 20_8 , and that the accumulator, AC, contains the value 10_8 . Describe in detail what happens when the PeANUt machine executes the following instruction that is in the current instruction register, CI.

```
mul x
```

Your answer should mention all the events that result in information flowing into any of the following components of the PeANUt machine, and the values held in the affected components:

PC, CI (Current Instr. Reg.), AC, XR, SP, EQ, GT, OV,
ALU, MAR (Mem. Address Reg.), MDR (Mem. Data Reg.).

[6 marks]

- (d) Write a sequence of PeANUt assembly language instructions that will result in `min` being set to the value of the smallest positive integer (value > 0) in the array `a`, where `a` is an array containing thirty (30) integers. If no positive integer is present in `a`, then the value of `min` is set to 0. Assume that `min` and `a` have already been defined, and that integer values have been stored in `a`. You may define and use temporary variables as required. Your program should be appropriately commented with C code. A portion of the marks will be awarded for clarity and economy of code size. [10 marks]

Question 4 [26 marks] More PeANUt

Consider the C program below.

```
int DivMod(int a, int b, int *r) {
    int q=0;
    if (b != 0)
        q = a/b;
    *r = a - q*b; /* = a%b */
    return (q);
} /* DivMod() */

main() {
    int x = 43, r, y; /* assume implemented as fixed memory locations */
    y = DivMod(x, 7, &r);
    ...
} /* main() */
```

(a) Assuming the procedure call convention of passing parameters and return values via the stack (as discussed in lectures) for PeANUt:

(i) Translate the call `y = DivMod(x, 7, &r);` of the above program into PeANUt assembly language. Define any macros that you use. [3 marks]

(ii) Draw the stack for the call in Part (a)(i) immediately before the execution of the first instruction of the `DivMod()` function. Assume that the address of the call `DivMod` instruction is decimal 40, and the address of `r` is decimal 32. Clearly indicate the cell that the stack pointer references and the contents of all cells in the stack. Label the cells with their respective symbolic names. [2 marks]

(iii) Give a PeANUt assembly language implementation of the `DivMod()` function of the above program. Give appropriate symbolic names to any stack offsets corresponding to parameters and return values, and define any macros that you use. [5 marks]

Question 4 (continued)

(b) Given the following PeANUt declaration:

```
n:      block   1      ;      short int n;
```

where **n** contains the value of a non-zero, positive integer in 16-bit 2's complement.

Write PeANUt assembly language code that leaves in **XR** the position of the most significant 1 bit in **n**. Note that bit positions range from 0 (least significant) to 15 (most significant). Declare any macros, variables and constants (eg. bit masks) that you use.

Example: if **n** contained the bit pattern 0010 1110 1001 0000, the value 13 should be left in **XR**. **[4 marks]**

(c) Assume that, on the PeANUt machine, the user program has established a (user-definable) trap number 16 to be handled by a procedure defined in the user program.

Explain how the operating system uses its trap table (**OSTT**) during the execution of a **trap #16** instruction. Also explain also how it manipulates the program counter register (**PC**) and the stack, so that the user program can return from the trap.

[3 marks]

(d) The following parts refer to the paging mechanism for implementing Virtual Memory.

(i) Consider an executing program (process) **P** which requires more virtual memory space than the physical memory size. Under what conditions can **P** execute with good paging behaviour, ie. execute with a relatively low number of page faults?

[2 marks]

(ii) Briefly explain why Least Recently Used (**LRU**) is generally a better paging policy than First In First Out (**FIFO**).

[2 marks]

(e) The following parts refer to multiprocessing.

(i) Give 1 advantage and 1 disadvantage of switching between processes upon I/O.

[2 marks]

(ii) What hardware mechanisms are required to implement *time-slicing*?

[1 marks]

(f) Briefly explain how *random access* files can be organized on a disk, so that any item (at a known position in the file) may be efficiently accessed. **[2 marks]**

Question 5 [24 marks] Further Topics and Related Material

Select any two (2) of the following topics, and write one to two pages on each of them. Clearly indicate which topics you are answering.

- (a) Describe five aspects of computer security. Which of these can public key cryptosystems provide? [12 marks]
- (b) In High Performance Computing, SIMD and MIMD are two approaches used to achieve parallelism. Describe these two approaches and the types of problems they are best suited to solving. [12 marks]
- (c) Describe the three layer model of computer communications. [12 marks]
- (d) The Internet enables packets of information to be transmitted from a source computer to a destination computer. Describe how this is achieved for computers on the same local network. How is it achieved for computers on adjacent (nearby) local networks? What about computers whose local networks are not adjacent? [12 marks]
- (e) Assume you are building a Beowulf-style cluster of workstations, such as the `bunyip`. Given you have a fixed amount of money to spend, what are the issues and trade-offs that need to be considered when designing the machine? [12 marks]
- (f) Give an analytic proof of the following statement, then describe an alternate approach using a super-computer to conclusively prove the same statement.

For all integers $n \geq 3$, there do not exist non-zero integers x, y , and z which satisfy $x^n + y^n = z^n$.

[12 marks]

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

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>f₀</i>	mode 0 (fixed)	implicitly specified by opcode
<i>f₁</i>	mode 1 (fixed)	implicitly specified by opcode
-	no mode is applicable	

<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 2: 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 3: Powers of 2 in decimal.