

THE AUSTRALIAN NATIONAL UNIVERSITY  
First Semester Examination – June 2004

**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 is linked.*

*The marking scheme will put a high value on clarity so, as a general guide, it is better to give fewer terse answers in a clear, succinct manner than to outline a greater number in a sketchy, half-answered fashion.*

*The Appendix contains information on the PeANUt instruction set.*

|                                   |
|-----------------------------------|
| Name (family name first):<br><br> |
|-----------------------------------|

|                         |
|-------------------------|
| Student Number:<br><br> |
|-------------------------|

*The following are for use by the examiners.*

|         |         |         |         |         |         |            |
|---------|---------|---------|---------|---------|---------|------------|
| Q1 Mark | Q2 Mark | Q3 Mark | Q4 Mark | Q5 Mark | Q6 Mark | Total Mark |
|         |         |         |         |         |         |            |

**Question 1 [15 marks] Digital Building Blocks**

- (a) Show how you would evaluate the following expression using a stack architecture with instructions *push*, *pop*, *add*, *sub*, *mul*, and *div*

$$(-1 + (2 * (3 - 4)) / 5 + 6 * 7) * 8$$

[2 marks]

- (b) Convert the *hexadecimal* (ie. base 16) number 749 to base 6. Show your workings.

[3 marks]

**Question 1 (continued)**

- (c) 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. What floating point value is represented by the following bit pattern?

11000010 11001010 00000000 00000000

[3 marks]

- (d)

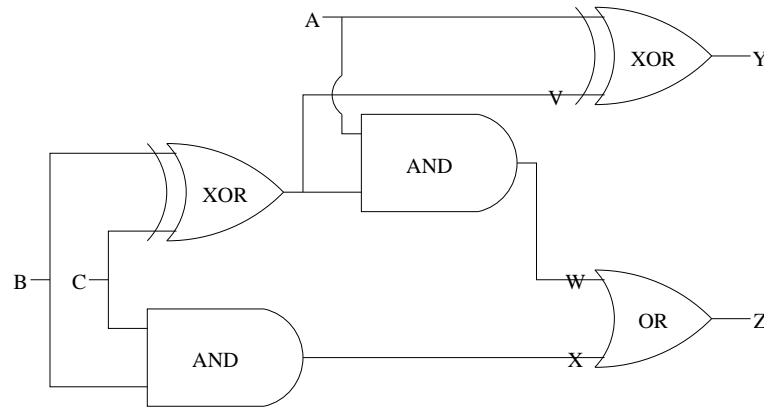
$$\begin{aligned}
 X &= -199 \text{ (decimal integer)} \\
 Y &= 1111\ 1100\ 1110\ 0110 \text{ (16 bit two's complement integer)} \\
 Z &= X - Y
 \end{aligned}$$

In the above what is the value of Z. Give your answer as both a 16 bit two's complement binary number AND as a decimal number. Show your working.

[4 marks]

**Question 1 (continued)**

- (e) The following circuit for a full-adder has been constructed using AND, OR and XOR (exclusive OR) gates. Each gate receives two input signals that are either 0 or 1, and produces a single output signal that is also either 0 or 1.



Complete the following truth table for the full-adder.

| A | B | C | V | W | X | Y | Z |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 |   |   |   |   |   |
| 0 | 0 | 1 |   |   |   |   |   |
| 0 | 1 | 0 |   |   |   |   |   |
| 0 | 1 | 1 |   |   |   |   |   |
| 1 | 0 | 0 |   |   |   |   |   |
| 1 | 0 | 1 |   |   |   |   |   |
| 1 | 1 | 0 |   |   |   |   |   |
| 1 | 1 | 1 |   |   |   |   |   |

[2 marks]

The two output channels for the full-adder are often referred to as the “sum” and the “carry out”. In the above which output channel is which? Which input channel is usually referred to as the “carry in”?

[1 mark]

**Question 2 [20 marks] C Programming**

(a) The following program examines two character strings using functions from the standard C library:

```
#include <stdio.h>
#include <ctype.h>

int main() {

    char name1[]="My-COMP2300-Notes\\";
    char name2[]="28-July-2003 \\noon\\";

    int i, nlower=-2, nalnum=-4;

    printf("C Programming Example\n" /* );
    printf("Comparison Strings" /* );
    printf("%s\n",name1);
    printf("%s\n",name2);

    for (i=0;i<15;i++){
        if (islower((int) name1[i]) || islower((int) name2[i])) nlower++;
        if (isalnum((int) name1[i]) && isalnum((int) name2[i])) nalnum++;
    }

    printf("Total number of lower case   =%5d\n",nlower);
    printf("Total number of alphanumeric =%5d\n",nalnum);

    return 0;
}
```

Relevant details from the `islower/isalnum` man page are given below:

**SYNOPSIS**

```
#include <ctype.h>

int islower (int c);
int isalnum (int c);
```

**DESCRIPTION**

```
islower()
    checks for a lower case charater [a-z]
isalnum()
    checks for an alphanumeric character [a-z:A-Z:0-9];
    it is equivalent to (ialpha(c) || isdigit(c)).
```

**RETURN VALUE**

The value returned is nonzero if the character `c` falls into the tested class, and zero if not.

**Question 2 (continued)**

(i) The program compiles and runs without errors. Write down exactly what output is given by this program. Number each line of the output and, if relevant, clearly indicate the presence of each blank space. The library functions produce no output, and there are less than 10 lines of output in total.

(i)

[4 marks]

(ii) Write C code for a function

```
int myislower(int c);
```

that performs exactly the same operations as the library routine `islower`. Your code **MUST** be less than 10 lines, and should **NOT** call any other function.

(ii)

[3 marks]

**Question 2 (continued)**

(iii) Write C code for a function

```
int myisalnum(int c);
```

that performs exactly the same operations as the library routine `isalnum`. Your code **MUST** be less than 15 lines, and should **NOT** call any other function.

(iii)

[4 marks]

(b) Every function in C has a definition and a prototype. What is the purpose of a function prototype?

[1 mark]

**Question 2 (continued)**

(c) The following code is contained in a file `strange.c`.

```
extern int result;
int mval = 0;

int strange(int* a, int* b)
{
    int tmp;
    static int count=0;
    count++;
    if (*a < *b){
        tmp = *b;
        *b = *a;
        *a = tmp;
    }
    mval = (*a > mval) ? *a : mval;
    result+=*a*b;
    return count;
}
```

(i) For each symbol given in the following table indicate if it will have a symbol table entry in the `.symtab` section of ELF file `strange.o`. If so, indicate whether the symbol type is `local`, `global`, or `extern`.

(i)

| Symbol  | strange.o .symtab entry? | Local/global or extern |
|---------|--------------------------|------------------------|
| result  |                          |                        |
| mval    |                          |                        |
| strange |                          |                        |
| tmp     |                          |                        |
| count   |                          |                        |
| a       |                          |                        |
| b       |                          |                        |

[3 marks]

(ii) You produce an executable load module by linking `strange.o` with the following main program

```
#include<stdio.h>
int strange(int* a, int* b);
int result;
int main()
{
    int i[]={1,2,3,4,5,6,7,8,9,10,11,12};
    int j, *p=i+3, final;
    final = strange(p,p+5); p++;
    final = strange(p,p+5); ++p;
    final = strange(p,p+5); p++;
    printf("Value of final = %5d\n",final);
    printf("Value of result = %-5d\n",result);
    printf("Array\n");
    for (j=0; j<12; j++)printf("%5d ",i[j]);
    printf("\n");
    return 0;
}
```

**Question 2 (continued)**

Exactly what is printed out when this executable is run. (Clearly mark all white space.)

(ii)

[3 marks]

(d) You compile and run the following C program on two different computers.

```
#include <stdio.h>
int mystery(void)
{
    int i = 9;
    return (*((char *) &i));
}
int main()
{
    printf("Value %d\n",mystery());
    return 0;
}
```

When executed on computer A the above program prints “Value 0”, while on machine B it prints “Value 9”. This is not due to any bug in the program. Explain what this code is doing and why it might report different values when run on different computers. What can you say about computer A and computer B?

[2 marks]

**Question 3 [35 marks] Assembly Level Machine Organization**

(a) (i) Which of the following four assembly instructions is not a valid PeANUt instruction? Explain in one sentence.

- A: `store @1023`
- B: `and !-1`
- C: `clov`
- D: `loadxr`

(i)

[1 mark]

(ii) What kind of error message would you expect the PeANUt assembler to print if you have the above non-valid instruction in your program?

(ii)

[1 mark]

(b) Why are *traps* used in PeANUt for input and output? Explain in one or two sentences and give a short example (a few lines of PeANUt assembly code) of how traps can be used to print the character '1' followed by a new line character.

[3 marks]

(c) What is the purpose of the *Accumulator* (AC) in PeANUt? Explain in one or two sentences.

[2 marks]

**Question 3 (continued)**

- (d) (i) Write a sequence of PeANUt 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[10];
int i = 1;
int x = 0;
int end = 10;

a[0] = 1;

while (i < end) {
    x = a[i-1];
    a[i] = x * 2;
    i = i + 1;
}
```

(i)

[8 marks]

- (ii) Write down the values in all elements of the array **a** after the program has been executed.

(ii)

[1 mark]

**Question 3 (continued)**

- (e) Assume the PeANUt accumulator contains the bit pattern **11001101 00001010** and at address **a10** (octal) in main memory the decimal value **510** is stored. The following two PeANUt instructions are executed:

**and a10**  
**not**

What bit pattern (binary value, 16 bits) is stored in the accumulator after these two instructions have been executed?

[1 mark]

- (f) Assume the accumulator **AC** contains a value of **5** (decimal), and at address **a3** (octal) the value **4** (decimal) is stored. The memory cell at address **a4** (octal) contains a value of **5**.

(i) Explain in detail what happens when the PeANUt machine executes the instruction **cmp @3**. Assume this instruction (**cmp @3**) is stored at address **a10**. Your answer should describe the events that happen in the phases *Fetch*, *Decode* and *Execute* and the values moved between the affected components **AC**, **CI**, **MAR**, **MDR** and **ALU**.

(i)

[4 marks]

(ii) Write down the values (in decimal) stored in the **AC**, the **PC**, the **MAR**, the **MDR**, and the flags **EQ**, **GT** and **OV** after the instruction **cmp @3** has been executed.

(ii)

[2 marks]

**Question 3 (continued)**

(g) Assume you have a Peanut assembly program that consists of a main program and a function `min()` that calculates and returns the minimum of three input parameters.

Assume the stack looks like given in the diagram on the right side below just after the function `min()` has been called. On the left side you can see parts of the corresponding Peanut assembly program.

In the comments to this code and with the stack diagram you find numbers in brackets `<1>` to `<9>`. Analyse the given code and the stack diagram and answer the questions (i) to (vii) below.

```
a:      data ... ; <1>
b:      data ... ; <2>
c:      data ... ; <3>
r:      block 1
```

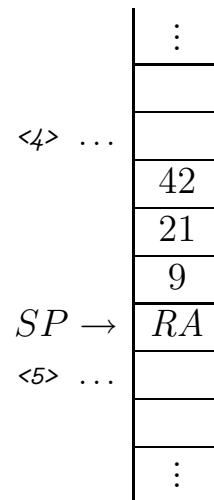
```
main:
    ... <6>
    call min
    ... <7> ; printf("%d", r);
           ; printf("\n");

min:
           ; int min(int a, int b, int c) {
           ;     int res;

    ... <8>

           ;     return res;
ret       ; }

    ... <9>
```



(i) Write down the values for the variables `a`, `b` and `c` that need to be defined at `<1>`, `<2>` and `<3>`.

(i)

[1 mark]

(ii) What will be stored in the memory cell at `<4>`? What numerical value will be stored?

(ii)

[1 mark]

**Question 3 (continued)**

(iii) What will the memory cell at <5> be used for?

(iii)

[1 mark]

(iv) Program the code that is needed before the call to the function `min()` at <6>. Follow the PeANUt procedure call convention. Comment your code.

(iv)

[2 marks]

(v) Program the code that is needed after the function at <7> to print the result and print a new-line character. Follow the PeANUt procedure call convention. Comment your code. Make sure you finish the main program properly.

(v)

[3 marks]

**Question 3 (continued)**

(vi) Program the function `min` at <8> according to the C comments given. It should return the minimum value of the three input parameters given. Follow the PeANUt procedure call convention. Comment your code.

(vi)

[3 marks]

(vii) What PeANUt assembly directive needs to be written at <9> so that this program can be assembled properly.

(vii)

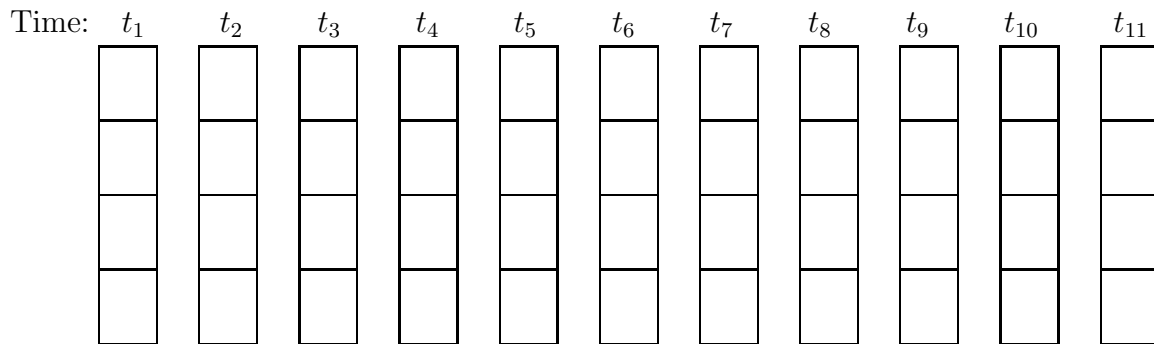
[1 mark]

**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 (LRU)* page replacement policy is implemented in 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>4</b> | <b>3</b> | <b>5</b> | <b>2</b> | <b>4</b> | <b>1</b> | <b>6</b> | <b>2</b> | <b>4</b> | <b>5</b> | <b>2</b> |

(i) Assuming that 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.



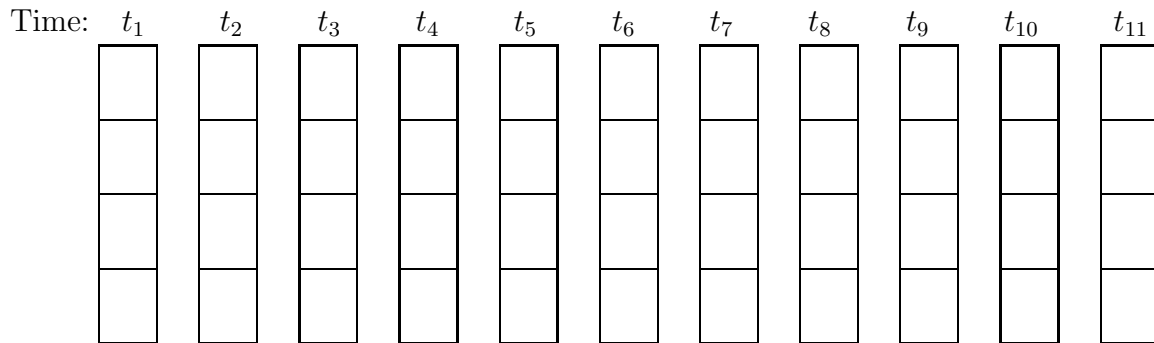
[3 marks]

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

(ii)

[1 mark]

(iii) Now assume the *first-in first-out (FIFO)* page replacement policy is used instead of LRU. Again assuming that the main memory has been empty at the beginning, complete the following diagram of the four main memory pages at times  $t_1$  to  $t_{11}$  for the same page access sequence as given above. Please write your answers – the page numbers – into the appropriate boxes.



[1 mark]

**Question 4 (continued)**

(iv) Is *FIFO* a better or a worse page replacement policy than *LRU* for the above page access sequence? Explain in one sentence.

(iv)

[1 mark]

(v) In general, which page replacement policy – *LRU* or *FIFO* – performs better? Explain in one or two sentences.

(v)

[1 mark]

(b) Cache memory systems are said to benefit from programs that show good *spatial* and *temporal* locality. In this context what is meant by spatial and temporal locality.

[2 marks]

(c) What is the difference between SRAM and DRAM? Name one advantage and one disadvantage of SRAM over DRAM.

[2 marks]

**Question 4 (continued)**

(d) What is firmware?

[1 mark]

(e) You are debugging your code on a Sun computer. The program counter is currently positioned at line20 of the following SPARC assembly code.

```

/*line20:*/.start: nop
/*line21:*/      mov 1,%10
/*line22:*/      add 2,%10,%10
/*line23:*/      ba  .foo
/*line24:*/      add 3,%10,%10
/*line25:*/      add 4,%10,%10
/*line26:*/.foo:
/*line27:*/      add 5,%10,%10
/*line28:*/      mov %10,%11

```

(i) You allow the program counter to advance to line28. At this point what is the value contained in register %10? Explain how you derive your answer.

(i)

[2 marks]

(ii) From executing line20 until immediately prior to executing line28 how many instructions are executed?

(ii)

[1 mark]

**Question 5 [10 marks] Operating System Concepts**

- (a) Operating systems share CPU resources between processes by timeslicing. What is a typical value (in seconds) for a timeslice on a multiuser computer? What might prompt a process to utilize the CPU for less than its default timeslice?

[2 marks]

- (b) What are the advantages of interrupt driven I/O over programmed I/O?

[2 marks]

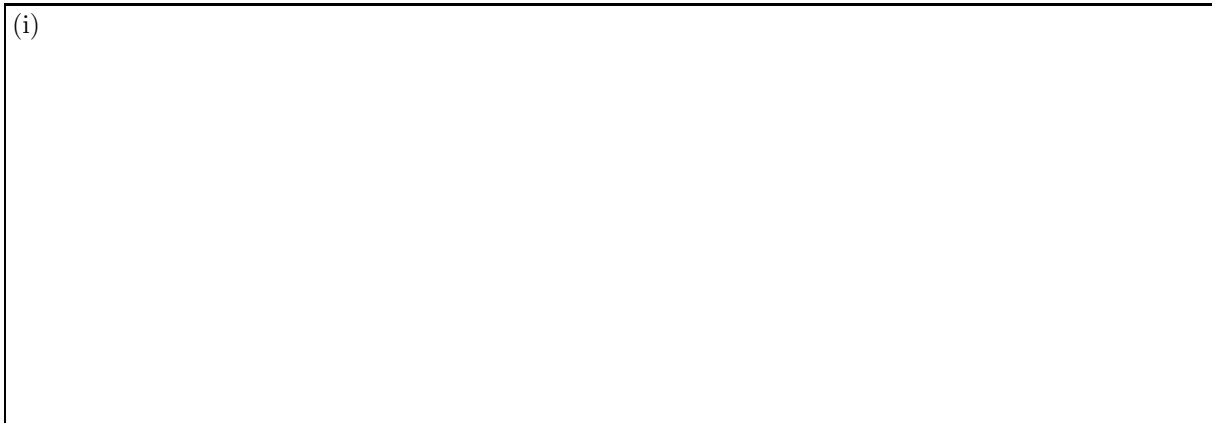
- (c) You purchase a new 15000RPM disk drive for your computer. The documentation states that the average seek time is 5ms and that on average there are 500 sectors per track. Based on this information what is the typical access time required to read one sector?

[2 marks]

**Question 5 (continued)**

(d) (i) Explain in two to three sentences how files are organised on a hard disk.

(i)



[2 marks]

(ii) What is file fragmentation on hard disks and why does it occur?



[2 marks]

**Question 6 [5 marks] Interconnection Networks**

(a) (i) What is meant by the terms local area network (LAN) and wide area network (WAN)?

(i)

[2 marks]

(ii) In constructing LANs and WANs redundancy is an important issue to consider. Explain how redundancy might be built into a network.

(ii)

[1 mark]

(b) Due to the finite speed of light the latency for a communication between two computers connected on a network increases as the physical separation increases.

(i) To the nearest power of 10 what is the value of the speed of light in m/s? What is the typical clock speed of a modern CPU? Use these values to determine how far light travels in 1 clock tick of a modern CPU.


(i)

[1 mark]

**Question 6 (continued)**

(ii) In addition to physical separation can you give two other factors that affect communication latency

(ii)



[1 mark]

Continuation of answer to Question  Part

Continuation of answer to Question  Part

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 <code>opcode</code> |
| $f_1$ | mode 1 (fixed)        | implicitly specified by <code>opcode</code> |
| -     | 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

| $x$ | $2^x$ |
|-----|-------|
| 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