

THE AUSTRALIAN NATIONAL UNIVERSITY
First Semester Examination – June 2001

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, a dictionary (English) for candidates for whom English is a second language

COMP2300: Answer all 5 questions.

ENGN2213: Answer questions 1, 2 and 3 only.

Questions are NOT equally weighted.

The questions are followed by labelled, framed blank spaces 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):

Student Number:

The following are for use by the examiners.

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

Question 1 [15 marks] Fundamental Concepts

- (a) Convert the decimal value 457_{10} into *hexadecimal* and *octal*. Show your workings.

Answer:
 $457_{10} = 1\ 1100\ 1001_2$
 $= 1C9_{16}$
 $= 711_8$

[2 marks]

- (b) What value is represented by the bit pattern $11111111\ 11010101$ if it is interpreted as a 16-bit two's complement integer? Give your answer in *binary* and *decimal*, showing your workings.

Answer:
Flip all bits left of the most right 1-bit.
 $1111\ 1111\ 1101\ 0101 \rightarrow 0000\ 0000\ 0010\ 1011 = 43_{10}$
Solution = -43_{10}

[2 marks]

Question 1 (continued)

- (c) Assume the 32-bit pattern 11000001 01100000 00000000 00000000 represents an *IEEE single precision floating-point* number. What is its decimal representation as a real number (e.g. like 42.13)? The IEEE single precision format is: 1 bit sign, 8 bits exponent with a bias of 127, and the remaining 23 bits are the mantissa. Show your workings.

Answer:
 Sign bit $s = 1 \rightarrow$ negative value
 Exponent $e = 1000\ 0010 = 128 + 2 = 130$
 $\rightarrow e = 130 - 127$ (bias) $= 3 \rightarrow 2^3 = 8$
 Mantissa: $m = (1).11 = 1 + 0.5 + 0.25 = 1.75$
 Solution: $-1 * 8 * 1.75 = -8 + 6 = -14.0$

[2 marks]

- (d) How is the 32-bit pattern 10101010 11001100 11110000 00000000 stored in four memory cells (each containing one byte) if the *Little-Endian* scheme for storing multi-byte values is applied? The figure below shows four memory cells with addresses 100 to 103. Fill in the bit patterns in the correct way.

	0000 0000	1111 0000	1100 1100	1010 1010	
	addr 100	addr 101	addr 102	addr 103	

[1 mark]

Question 1 (continued)

- (e) Which processor family uses the *Big-Endian* scheme to store multi-byte values?

Solution (Big-Endian processors): **SPARC, Motorola 680xx,**
PowerPC and MIPS (which are both Big- and Little!)

[1 mark]

- (f) List (in one sentence each) three architectural differences between basic RISC and CISC.

Examples include

- RISC is **load/store**, CISC has many **various addressing modes**
- RISC instructions are **simple**, CISC are **complex**
- RISC offers only **basic instructions**, CISC many **specialised instructions**
- RISC instruction take **one cycle**, CISC one or **more**
- RISC instructions are **hardwired**, CISC instructions are **microprogrammed** (RISC hardware generally simpler than CISC hardware)
- Other ideas

[3 marks]

- (g) Define in one sentence the function of the *Program Counter*.

Answer:

Points to the address in memory of **next (or current) instruction** to be fetched.

[2 marks]

- (h) Define in one sentence the function of the *Bus*.

Answer:

Communication link(s) between the main components of a computer.

[2 marks]

Question 2 [20 marks] C Programming Language

- (a) Complete the following C code so that it prints the maximum value stored in the array `a` of positive integers. You can assume that the function `ReadValues` (which reads `N` numbers from keyboard and stores them in the array `a`) is provided, so you don't have to program it. Only program the part which is missing. A good solution requires less than 10 lines of C code.

```
#include <stdio.h>
#define N 10 /* Length of the array */

int main(void) {
    unsigned int a[N];
    int max;      /* Contains maximal value at the end */
    int i;        /* Loop counter */

    ReadValues(a, N); /* Read values from keyboard */
    /* The array a now contains N positive numbers */

    /* ... your code goes here ... */

    /* max should now contain the maximum value from a */
    printf("max=%d\n", max);

    return 0;
} /* End main */
```

Solution:

```
max = a[0];
for (i=1; i<N; i++) {
    if (a[i] > max) {
        max = a[i];
    }
}
```

Alternative:

```
max = -1;
for (i=0; i<N; i++) {
    ...
}
```

[8 marks]

Question 2 (continued)

(b) Below you can see two different implementations of a function `reverse` that are claimed to reverse the given input string `s`.

```
void reverse(char s[]) {
    int i, j;
    int len=0;
    char tmp;

    do {len = len+1;}
    while (s[len] != '\0');
    i = 0;
    j = len-1;
    while (i != j) {
        tmp = s[i];
        s[i] = s[j];
        s[j] = tmp;
        i = i+1;
        j = j-1;
    }
}
```

```
void reverse(char s[]) {
    int i, j;
    int len=0;
    char tmp;

    while (s[len] != '\0') {
        len=len+1;
    }
    j = len / 2;
    for (i=0; i<j; i++) {
        tmp = s[i];
        s[i] = s[len-1-i];
        s[len-1-i] = tmp;
    }
}
```

Which one of the two functions is correct? What is wrong with the other function? Explain.

Answer:

The function on the **left is wrong** because:

- Variable `len` is set to 1 if the string `s` has length 0.
- `while` loop doesn't stop if length of the string `s` is even, because `i` and `j` cross over. Ultimately the program crashes.

[6 marks]

Question 2 (continued)

(c) Have a look at the following C program:

```
#include <stdio.h>

int do_it(int a, int b, int c) {
    if (a > b) {
        b = a;
    }
    if (c < b) {
        return b;
    }
    return c;
}

int main(void) {
    int w, x, y, z;

    w = 9;
    x = 4;
    y = 7;
    z = 3;

    w = do_it(x, y, z);
    printf("w=%d\n", w);
}
```

What is the function `do_it` doing? Explain briefly.

Answers:

The function `do_it` returns the maximum value of `a`, `b`, and `c`.

[2 marks]

What is the output of the program? Explain briefly.

The output is `w=7`, because `w` is overwritten with the result of `do_it`.

[2 marks]

Question 2 (continued)

- (d) Which of the following `printf` statements will print out the value of the floating-point variable `myfloat`? Give as answer either A, B, C, D, E or F.

A: `printf("%d", myfloat);` C: `printf("%c", myfloat);` E: `printf("%f", &myfloat);`
B: `printf("myfloat");` D: `printf("%f", myfloat);` F: `printf("%d", &myfloat);`

D

[1 mark]

- (e) Which of the following `scanf` statements do you have to use to read a string of characters into the array `myword`? Give as answer either A, B, C, D, E or F.

A: `scanf("%d", &myword);` C: `scanf("%c", myword);` E: `scanf("%s", myword);`
B: `scanf("%s", &myword);` D: `scanf("%s\n", &myword);` F: `scanf("%f", myword);`

E

[1 mark]

Question 3 [25 marks] PeANUt

- (a) Give (in one sentence each) two reasons why symbolic names for addresses (labels and variables) are useful in assembly language (as compared with machine language, where all addresses have to be 'hard-coded' into a program).

Example answers:

1. Code is **relocatable** (no fixed addresses for variables, jumps, branches, labels, etc.)
2. It is **easier to change code**, no problem with inserting or deleting instructions (no need to change addresses for labels, jumps, branches etc.)
3. Code is much **more readable**

[2 marks]

- (b) What is the main purpose of the *Index Register* (XR)? Explain in one or two sentences and give a short example (two or three lines of PeANUt assembly code) of how it can be used to load the fifth element in an integer array *a* (assuming indices start with 0) into the accumulator.

Answer:

- Mainly used to store an **index** number to access elements in an **array**.
- Used with **indexing addressing mode**, where the value of the index register is added to the operand.
- Code example:

```
load #4
storexr
load *a
```

[3 marks]

- (c) Which of the following PeANUt assembly instructions do you have to use to add the value which is stored at memory address *a07* to the value which is in the accumulator? Give as answer either A, B, C, D, E or F.

A: add #07
B: load 07

C: load #07
D: add @07

E: add \$07
F: add 07

F

[1 mark]

Question 3 (continued)

- (d) Suppose you want to clear (set to zero) the upper 8 bits stored in the PeANUt accumulator. Which of the following instructions would you use? Give as answer either A, B, C, D, E or F.

A: and %0000000011111111 C: and #255 E: and 255
B: and #256 D: and #%0000000011111111 F: and #%1111111100000000

None are correct! Foolish examiner got this wrong!

[1 mark]

- (e) Write a sequence of PeANUt assembly language instructions that do the same as the following piece of C code. Use trap #1 to end your program.

```
int f1, f2, i, end, tmp;

f2 = 0;
f1 = 1;
tmp = 0;
i = 2;
end = 42;

while (i < end) {
    tmp = f1 + f2;
    f2 = f1;
    f1 = tmp;
    i = i + 1;
}
```

```
f1:   block   1   ; Alternative: f1: data 1
f2:   block   1   ; Alternative: f2: data 0
i:    block   1   ; Alternative: i:  data 2
end:  block   1   ; Alternative: end: data 42
tmp:  block   1   ; Alternative: tmp: data 0

      load #0      ; If 'data' is used this initialisation
      store f2     ; can be obmitted
      store tmp
      load #1
      store f1
      load #2
      store i
      load #42
      store end   ; Alternative:
loop: load end     ; load i
      cmp i        ; cmp end
      ble stop    ; beq stop
      load f1
      add f2
      store tmp
      load f1
      store f2
      load tmp
      store f1
      load i
      add #1
      store i
      jmp loop
stop: trap #1
```

[9 marks]

Question 3 (continued)

- (f) Assume the accumulator (AC) contains the value 42_8 , and that memory cell 100_8 contains the value 24_8 . Describe in detail what happens when the PeANUt machine executes the following instruction which is stored at memory address 10_8 .

add 100_8

Your answers should describe the events that happen in the three phases *Fetch*, *Decode* and *Execute*. Give the values moved between or held in the affected components:

AC, PC, CI, MDR, MAR, ALU, EQ, GT, OV

Fetch: The instruction add is fetched (loaded) from main memory and copied via the MDR into the CI register.

MAR \leftarrow PC
 Memory read, enable
 MDR \leftarrow Mem[MAR]
 CI \leftarrow MDR

Decode: The 16 bits in CI are decoded, and the operand (CI[9..0]) is copied into the MAR, so the operand value can be loaded from memory. The ALU is set to perform an addition.

MAR \leftarrow CI[9..0]
 ALU add

Execute: The operand is loaded from memory and the ALU performs the addition. The result is copied back into the accumulator.

Memory read, enable
 MDR \leftarrow Mem[MAR]
 AC \leftarrow AC + MDR

[9 marks]

Question 4 [20 marks] More PeANUt (COMP2300 only)

- (a) The PeANUt program below consists of a main program and a function myfunct. In the comments to this code you find numbers <1> to <8>. Analyse the code and answer the questions below.

```
a = ... ; <1>
b = ... ; <1>
c = ... ; <1>
RV = ... ; <1>

myfunct: load !a ; <3> int myfunct(int a, int b, int c) {
        cmp !b
        bgt label1
        load !b
label1:  cmp !c
        bgt label2
        load !c
label2: store !RV ; <7>
        ret ; }

main:   incsp #1 ; <5> main() {
        load #1
        incsp #1
        store !0
        load #6
        incsp #1
        store !0
        load #3
        incsp #1
        store !0 ; <2>
        call myfunct
        incsp #... ; <4>
        load !0
        incsp #-1
        add #48 ; <6>
        trap #3
        load #10 ; <8>
        trap #3 ; <8>
        trap #1

        end main
```

The following questions refer to the PeANUt assembly code above.

Give the values for the four concise macros at <1>.

```
a = -3
b = -2
c = -1
RV = -4
```

[2 marks]

Question 4 (continued)

How does the stack look like after the instruction at <2> and the instruction at <3> has been executed, respectively. Draw two diagrams of the stack, assuming the stack pointer (SP) has been pointing to address 100 at the beginning of the main program. Show the contents of the stack, and where the stack pointer points to in both cases.

- After instruction <2>:
a100: empty, a101: (RV), a102: 1 (a), a103: 6 (b), a104: 3 (c), SP points to a104
- After instruction <3>:
a100: empty, a101: (RV), a102: 1 (a), a103: 6 (b), a104: 3 (c), a104: (RA), SP points to a105

Also requires a nice diagram!

[6 marks]

By which value has the stack pointer to be changed at the instruction <4>?

-3

[1 mark]

What is the purpose of the instruction at <5>?

Make a 'free' memory cell for the return value (RV).

[1 mark]

What is the purpose of the instruction at <6>?

Convert integer value in corresponding ASCII value.

[1 mark]

Question 4 (continued)

What value will be stored in RV at instruction <7>?

6 will be stored in RV.

[1 mark]

What is the purpose of the two instructions at <8>?

Print a new line (return).

[1 mark]

What is the function `myfunct` doing?

`my_funct` returns the maximal value of `a`, `b` and `c`

[1 mark]

(b) What are traps good for ?

- Perform crucial operations within the operating system.
- Hide details in operating system, abstraction is visible by user.
- Protect/restrict access to operating system.
- Catch errors and failures.
- other suggestions

[3 marks]

(c) Why do modern operating systems support virtual memory?

Answers:

- Allow programs that need more memory than physically available.
- Needed because of memory hierarchy: Lots of cheap slow memory, but it's not possible to have large amounts of fast memory.
- Other suggestions

[3 marks]

Question 5 [20 marks] Further Topics (COMP2300 only)

(a) What is a network *Service Access Point*?

Answer: Access point for applications on a computer. Communication happens between computers, that are addressed by an address and a SAP identifier. On the Internet/Unix, SAP are port numbers, like 80 for web servers, 23 for Mail, etc. Various protocols are used for communication between applications.

[3 marks]

(b) What is a main disadvantage of a LAN with bus topology?

Answers:

- The bus becomes a bottleneck if more than one node wants to communicate (send).
- Packet sniffing is possible, as data is sent to all nodes.
- Other

[2 marks]

(c) Describe the technical limitations of a sequential computer.

Answers:

- Processor speed
- Input/Output bandwidth
- Memory size and bandwidth

[3 marks]

(d) What kind of parallel programming technique would you use on the ANU *Bunyip*?

Answers:

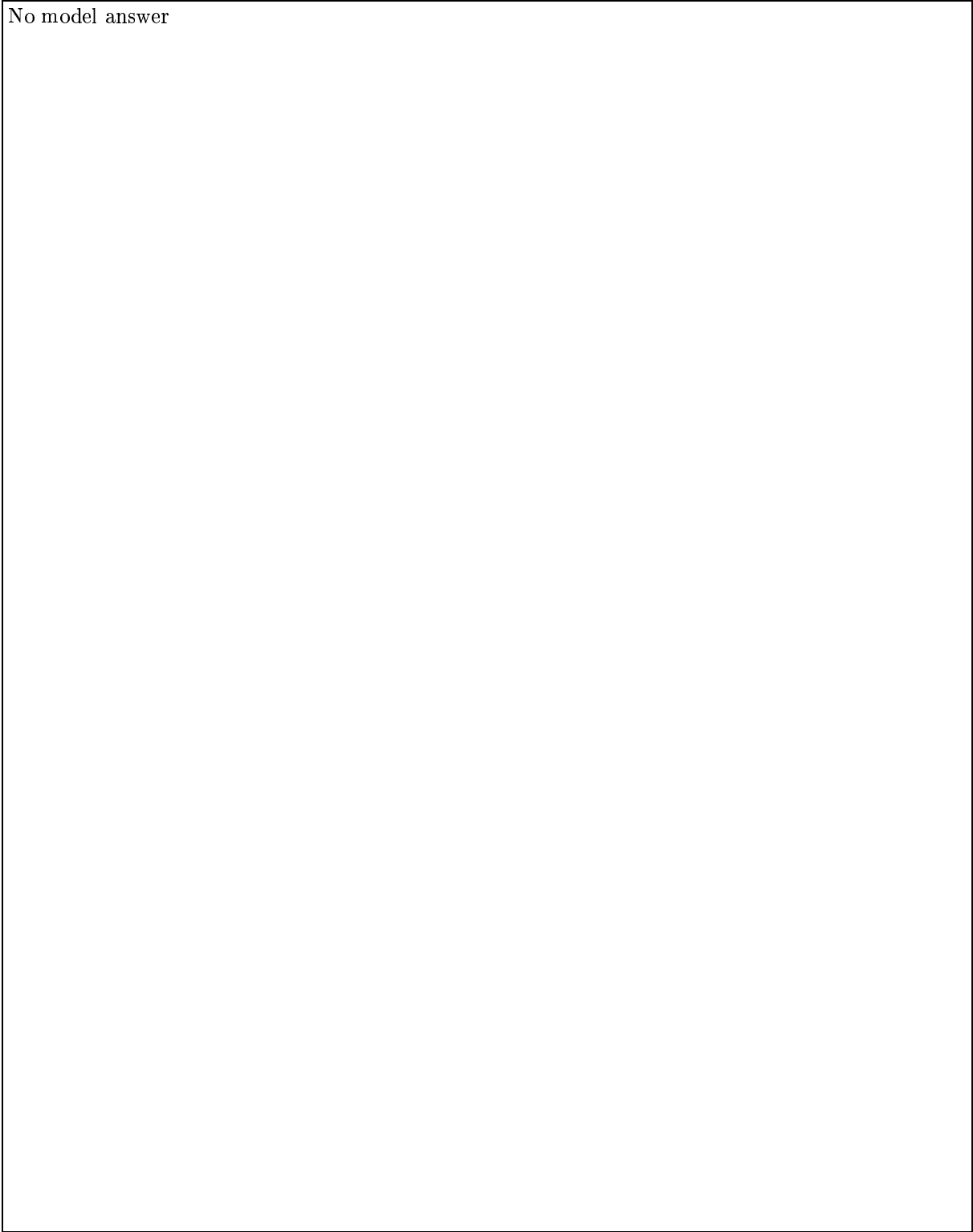
- Message passing, i.e. sending data from one node to the other via messages over the network.
- Use communication libraries like MPI or PVM.
- other

[2 marks]

Question 5 (continued)

- (e) Select one of the following topics and write one page on it. Clearly indicate which topic you are answering.
 - (A) The three layer model of computer communications.
 - (B) Architectural differences of shared vs. distributed memory architectures.
 - (C) Network routing in the Internet.

No model answer



[10 marks]

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 opcode
f_1	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

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