Computer Organization - Assignment 1

School of Electrical Engineering, Tel Aviv University

Instructor - Shai Avidan

TAs -Yoav Chachamovitz, Yoav Yosif Or

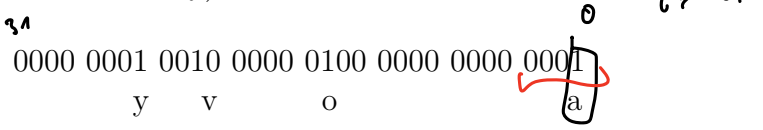
December, 2024

- The assignment consists of **two** questions.
- Write the solutions in the space provided below the relevant section. If you need extra space, please use the back of the page.
- A sheet describing the MIPS ISA will be provided to you during the quiz.
- Please make sure your code is clear and readable.

1 Question

1.1 Hashing in C

You are given a 4-letter word (e.g., "yoav") and need to map it to a 32-bit word as follows: The letter 'a' will be mapped to bit 0, the letter 'b' will be mapped to bit 1, etc. For example, the word "yoav" will be mapped to the following 32 bits (broken into 8 groups of 4 bits for better readability):



We show below the location of the letters in the word "yoav", for reference only.

You are given a C implementation below. However, **one** of the code lines are missing. Please fill them out. You can assume that there are no repetitions and that all characters are lowercase (ASCII 97('a')-122('z')), in the string "`word`".

```
char word[4];
int hash_int = 0x0000;        /*initialize the 32-bit word*/
for (i=0;i<4;i++)
{
    k = (int)word[i] - 97;          /*97 := 'a' in ASCII*/
    hash_int += 0x0001<<k


};
```

1.2 C to MIPS

The above code, converted to MIPS assembly is written below, with some blank lines. Fill them out to correctly execute the code. The **addresses** of the string `word` and `hash_int` are stored in `$s0` and `$s1` respectively.

Label	Instruction	Comment
	addi $t0 $zero 0	#initialize `hash_int`
	addi $t1 $zero 0	#initialize "i"
	addi $t2 $zero 1	#store constant 1
FOR:	addi $t3 $t1 -4	#$i = i - 4$
	beq $t3 $zero END	if i=4, break (go to END)
	add $t3 $t1 $s0	$t3 holds adress of i-th byte
	(load byte) lb $t3 0($t3)	#load `word[i]`
	addi $t3 $t3 -97	calculating k
	sllv $t3 $t2 $t3	#shift left by `k`
	add $t0 $t0 $t3	#change the correct bit
	addi $t1 $t1 1	inc
	j FOR	#jump to label FOR
END:	sw $t0 0($s1)	update the hash-int

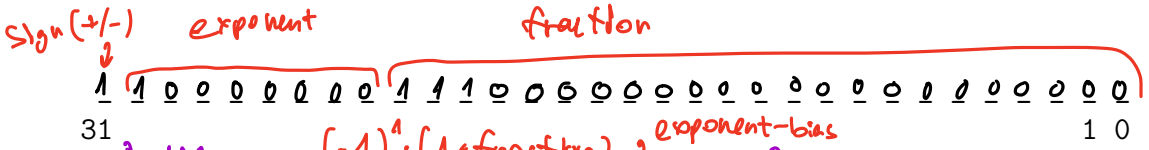
2 Question

2.1 Introduction

For this question please read about Fast Inverse Square Root Algorithm:
`https://en.m.wikipedia.org/wiki/Fast_inverse_square_root`

2.2 Float representation

Give the float representation of the number $x = -3.75$ using the IEEE 754-1985 32-bit floating-point specification.



2.3 C Code

Consider the following code and complete the assignment of y to i (the line after the assignment of *number* to y).

$\text{exponent} = \lfloor \log_2 3.75 \rfloor + 127$

$\text{fraction} \Rightarrow 3.75_{10} \rightsquigarrow 1.11 \cdot 2^1$

$= 3_{10} + 0.75_{10}$

$= 11_2 + 0.11_2$

$= 11.11_2$ $\left(1 \cdot 2^{-1} + 1 \cdot 2^{-2} = \frac{1}{2} + \frac{1}{4} = 0.75\right)$

$(\text{sign}) = 1.11 \cdot 2^1$

$-1 = 0.11_2 \cdot 1$

```
float Q_rsqrt( float number )
{
    long i;
    float x2, y;
    const float threehalfs = 1.5F;

    x2 = number * 0.5F;
    y  = number;
    i  = *( long * ) &y;
    i  = 0x5f3759df - ( i >> 1 );
    y  = * ( float * ) &i;
    y  = y * ( threehalfs - ( x2 * y * y ) );   // 1st iteration

    return y;
}
```

evil floating point bit level
what???

2.4 Approximation

The algorithm assumes that since $m_x \in [0, 1)$, the logarithm $log_2(1 + m_x)$ can be approximated by (fill in the right hand side of the equation below):

$$log_2(1 + m_x) \approx m_x + \sigma \tag{1}$$

σ is a free parameter

2.5 Derivation

Complete the missing line (the second row) in the following derivation:

$$\begin{aligned} I_x &= E_x L + M_x \\ &= L(e_x + B + m_x) \\ &= L(e_x + m_x + \sigma + B - \sigma) \\ &\approx Llog_2(x) + L(B - \sigma) \end{aligned} \tag{2}$$

where I_x is an integer interpretation of the floating-point bit-pattern of the float number x, $L = 2^{23}$, m_x is the binary representation of the significand, $E_x = e_x + B$ is the biased exponent (where $B = 127$), and $M_x = m_x * L$.

2.6 Newton's method

In Newton's method, if there is an approximation y_n for y, then a better approximation y_{n+1} can be calculated by (fill in the equation below):

$$y_{n+1} = \frac{y_n(3 - xy_n^2)}{2} \tag{3}$$