**Question #1** (4 points)

Let’s consider the MIPS64 processors seen during lectures.

You are requested to describe:

1. The general characteristics of MIPS64 processors;
2. What are RISC and CISC architectures and their advantages and disadvantages;
3. What are the instruction formats of the MIPS64 processors;
4. The difference between a MIPS64, a MIPS32 and a MIPS16 processor.

Write your answer here.

**Question 2** (4 points)

Let's consider a MIPS64 pipelined architecture including the following functional units (for each unit the number of clock periods to complete one instruction is reported):

* Integer ALU and Data memory: 1 clock period;
* FP arithmetic unit: 2 clock periods (pipelined);
* FP multiplier unit: 3 clock periods (pipelined);
* FP divider unit: 6 clock periods (unpipelined);

You should also assume that:

* The branch delay slot corresponds to 1 clock cycle, and the branch delay slot is not enabled;
* Data forwarding is enabled;
* The EXE phase can be completed out-of-order.

You should consider the following code fragment and, filling the following tables, determine the pipeline behavior in each clock period, as well as the total number of clock periods required to run it.

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* C \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; for (i = 30; i > 0; i--) {

v5[i] = (v1[i]/v2[i])\*(v3[i]/v4[i])–v2[i]+v4[i];

; }

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* MIPS64 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

|  |  |  |
| --- | --- | --- |
| .data | Comments | Clock cycles |
| v1: .double “30 values” |  |  |
| v2: .double “30 values” |  |  |
| v3: .double “30 values” |  |  |
| v4: .double “30 values” |  |  |
| v5: .double “30 values” |  |  |
|  |  |  |
| .text |  |  |
| main: daddui r1,r0,0 | r1 ← pointer |  |
| daddui r2,r0,30 | r2 ← 30 |  |
| loop: l.d f1,v1(r1) | f1 ← v1[i] |  |
| l.d f2,v2(r1) | f2 ← v2[i] |  |
| div.d f5,f1,f2 | f5 ← v1[i] / v2[i] |  |
| l.d f3,v3(r1) | f3 ← v3[i] |  |
| l.d f4,v4(r1) | f4 ← v4[i] |  |
| div.d f6,f3,f4 | f6 ← v3[i] / v4[i] |  |
| sub.d f7,f7,f2 | f7 ← –v2[i] |  |
| add.d f7,f7,f4 | f7 ← –v2[i] + v4[i] |  |
| mul.d f5,f5,f6 | f5 ← f5 \* f6 |  |
| add.d f5,f5,f7 | f5 ← f5 + f7 |  |
| s.d f5,v5(r2) | v5[i] ← f5 |  |
| daddi r2,r2,-1 | r2 ← r2 – 1 |  |
| daddui r1,r1,8 | r1 ← r1 + 8 |  |
| bnez r2,loop |  |  |
| halt |  |  |
| Total: |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| main: daddui r1,r0,0 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| daddui r2,r0,30 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| loop: l.d f1,v1(r1) |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| l.d f2,v2(r1) |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| div.d f5,f1,f2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| l.d f3,v3(r1) |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| l.d f4,v4(r1) |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| div.d f6,f3,f4 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| sub.d f7,f7,f2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| add.d f7,f7,f4 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| mul.d f5,f5,f6 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| add.d f5,f5,f7 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| s.d f5,v5(r2) |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| daddi r2,r2,-1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| daddui r1,r1,8 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| bnez r2,loop |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| halt |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

**Question 3** (6 points)

A 8x8 matrix MATR of bytes stores A-Z ascii characters only (all in upper case; English alphabet). Write an 8086 assembly program which extracts the characters where i=j (main diagonal) and counts how many occurrences of these characters are found in all the matrix. In other words, the program needs to fill in the two arrays KODE and OCCURRENCES according to the following rule:

KODE (k) = MATR (k,k) (i.e., the ascii code of the character stored in that position).

OCCURRENCES (k) = number of times the character KODE (k) is found inside the matrix MATR

Please observe/comply with the following

* It is mandatory to cut the matrix by rows.
* In your solution, please provide the declaration of all the arrays and the code, together with a short description of the algorithm used and significant comments to the code and instructions.
* It is guaranteed that MATR only stores A-Z ascii English alphabet characters (all in upper case) and that the characters on the main diagonal are all different.
* As this is an assembly program, please do NOT design an algorithm which is suitable to a high-level language approach, but strongly focus on the cut by rows of the matrix and its related properties. (= refer to its array implementation and “do not use” the original i and j).
* ANY (EVEN PARTIAL) BRUTE FORCE APPROACH IS NOT ACCEPTABLE. Any high-level-language-like approach is discouraged; please look at the array implementation!
* Hint: to devise a suitable algorithm, take as an example a smaller matrix (e.g. 4x4), “write it” when cut by rows, and identify the property of elements on the main diagonal.

Example:

Matrix MATR

C D A F K K J M

B B B D H G R E

O O P U Y R E F

W W W W F R Y Z

T T T T T T T T

D E A H T U I O

R E R T S W E T

B T U O K Z X D

KODE = C B P W T U E D

OCCURRENCES = 1 4 1 5 12 3 5 4

**Write your code in a file saved in the 8086 folder.**

Click on the following link to open a web page with the 8086 instruction set:

<http://www.jegerlehner.ch/intel/IntelCodeTable.pdf>

**Question 4** (9 points)

The IEEE-754 SP standard expresses floating-point numbers in 32 bits:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 31 | 30 | 23 | 22 | 0 |
| sign | exponent | | mantissa | |

Bit 31 is 0 if the number is positive, 1 if negative.

Write the addFPpositiveNumbers subroutine, which receives in input two 32-bit numbers, considers them as IEEE-754 SP floating point numbers, and returns their sum (in the same format). Bit 31 of the two input numbers is always 0 (i.e., the two numbers are positive).

In details, the subroutine implements the following steps:

1. take the mantissa of the two parameters
2. set bit 23 of the mantissa to 1
3. compare the two exponents. If they are equal, the exponent of the result is the same. If they are different:
   1. the exponent of the result is the highest one
   2. shift right the mantissa of the number with the lower exponent by as many position as the difference between the two exponents.
4. sum the two mantissas: this is the mantissa of the result. If bit 24 of the mantissa of the result is 1:
   1. shift right the mantissa of the result by one position
   2. increment the exponent of the result by one.
5. set bit 23 of the mantissa of the result to 0.
6. combine the mantissa and the exponent to get the final result.

Example: parameter1 = 0100 0010 0100 1011 0000 0000 0000 0000

parameter2 = 0100 0001 1010 0100 0000 0000 0000 0000

1. mantissa1 = 0000 0000 0100 1011 0000 0000 0000 0000   
   mantissa2 = 0000 0000 0010 0100 0000 0000 0000 0000
2. mantissa1 = 0000 0000 1100 1011 0000 0000 0000 0000   
   mantissa2 = 0000 0000 1010 0100 0000 0000 0000 0000
3. exponent1 = 1000 0100  
   exponent2 = 1000 0011
   1. exponentResult = 1000 0100
   2. mantissa2 = 0000 0000 0101 0010 0000 0000 0000 0000
4. mantissaResult = 0000 0001 0001 1101 0000 0000 0000 0000
   1. mantissaResult = 0000 0000 1000 1110 1000 0000 0000 0000
   2. exponentResult = 1000 0101
5. mantissaResult = 0000 0000 0000 1110 1000 0000 0000 0000
6. result = 0100 0010 1000 1110 1000 0000 0000 0000

Important notes:

1. **Create a new project with Keil inside the “ARM” directory and write your code there. The “ARM” directory contains some subdirectories that you can add to your project if you need them.**
2. In the Keil project, the following settings apply to the LPC1768 board:
   * Simulator: dialog DLL DARMP1.DLL, parameter -pLPC1768
   * ULINK2/ME Cortex Debugger: dialog DLL TARMP1.DLL, parameter -pLPC1768
3. The assembly subroutine must comply with the ARM Architecture Procedure Call Standard (AAPCS) standard (in terms of parameter passing, returned value, callee-saved registers).
4. Click on the following links to open web pages with the ARM instruction set

<http://www.keil.com/support/man/docs/armasm>

<https://developer.arm.com/documentation/ddi0337/e/Introduction/Instruction-set-summary?lang=en>

1. You can use the following link to convert between the IEEE 754 SP notation and the corresponding value:

<https://www.h-schmidt.net/FloatConverter/IEEE754.html>

**Question 5** (5 points)

Add a C file (e.g. sample.c) to the project created in the previous exercise.  
Write here the main function (which needs to be called from the Reset handler).  
Inside the main function, call the addFPpositiveNumbers subroutine, passing two floating-point numbers. If the result is higher than 3.1415, you have to switch on led 4 and switch off all other leds. Otherwise, switch on led 5 and switch off all other leds.