**Question #1** (4 points)

Let’s consider the VLIW processors seen during lectures.

You are requested to describe:

1. What is binary compatibility;
2. What they are and their basic characteristics;
3. Their advantages and disadvantages, in detail;
4. What are the three main tasks that classify the VLIW processors with respect to Superscalar and EPIC processors.

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;
* Memory Access (MEM stage) for Load/Store instructions: 2 clock periods;
* FP Arithmetic Unit: 2 clock periods (pipelined);
* FP Multiplier Unit: 4 clock periods (pipelined);

You should also assume that:

* The branch delay slot corresponds to 1 clock cycles, 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 = 0; i < 30; i++) {

v6[i] = - v4[i] + v5[i] + (v1[i] \* v2[i]) \* v3[i+1];

; }

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

|  |  |  |
| --- | --- | --- |
| .data | Comments | Clock cycles |
| v1: .double “30 values” |  |  |
| v2: .double “30 values” |  |  |
| v3: .double “31 values” |  |  |
| v4: .double “30 values” |  |  |
| v5: .double “30 values” |  |  |
| v6: .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] |  |
| mul.d f6,f1,f2 | f6 ← v1[i] \* v2[i] |  |
| daddui r3,r1,8 | r3 ← r1 + 8 |  |
| l.d f3,v3(r3) | f3 ← v3[i+1] |  |
| mul.d f6,f6,f3 | f6 ← f6 \* v3[i+1] |  |
| l.d f4,v4(r1) | f4 ← v4[i] |  |
| sub.d f6,f6,f4 | f6 ← f6 – v4[i] |  |
| l.d f5,v5(r1) | f5 ← v5[i] |  |
| add.d f6,f6,f5 | f6 ← f6 + v5[i] |  |
| s.d f6,v6(r2) | v6[i] ← f6 |  |
| daddui r1,r1,8 | r1 ← r1 + 8 |  |
| daddi r2,r2,-1 | r2 ← r2 – 1 |  |
| 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) |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| mul.d f6,f1,f2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| daddui r3,r1,8 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| l.d f3,v3(r3) |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| mul.d f6,f6,f3 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| l.d f4,v4(r1) |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| sub.d f6,f6,f4 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| l.d f5,v5(r1) |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| add.d f6,f6,f5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| s.d f6,v6(r2) |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| daddui r1,r1,8 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| daddi r2,r2,-1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| bnez r2,loop |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| halt |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

**Question 3** (6 points)

An 8 x 8 matrix MATR of bytes stores ASCII CHARACTER “0” digits in its border cells and “1” to “9” ASCII CHARACTER digits in all the other internal cells. The matrix is cut by rows and implemented by the array FIELD. Write an 8086 program which, having received an index k of FIELD, computes a value to be stored in in RES DB ? as follows:

* If FIELD[k] is a border cell (i.e. if FIELD[k]=”0”), then the program returns the value 0 in RES
* Otherwise, the program returns in RES the sum of the corresponding values of the cells which are its immediate vertical and horizontal neighbors in MATR; (the value of the cell should not be added).

c

a \* b

d

Please observe/comply with the following

* It is mandatory to cut the matrix by rows.
* In your solution, please provide the declaration of FIELD 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 is compliant with the requirements written above.
* 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 FIELD 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!
* **DO NOT WRITE A GENERAL-PURPOSE PROGRAM, BUT A SPECIFIC ONE TO SOLVE THIS PROBLEM.**

Example of Matrix MATR

“0” “0” “0” “0” “0” “0” “0” “0”

“0” “5” “2” ““3”” “4” “5” “7” “0”

“0” “4” “2” “2” “2” “2” “2” “0”

“0” “6” “5” “4” “9” “8” “9” “0”

“0” “7” “4” “1” “2” “4” “7” “0”

“0” “9” “9” “9” “9” “9” “9” “0”

“0” “4” “5” “6” “7” “6” “5” “0”

“0” “0” “0” “0” “0” “0” “0” “0”

FIELD = “0”, “0”, “0”, “0”, “0”, “0”, “0”, “0”, “0”, “5”, “2”, “3”, “4”, “5”, “7”, “0”, “0”, “4”, “2”, “2”, “2”, “2”, “2”, “0”, “0”, “6”, “5”, “4”, “9”, “8”, “9”, “0”, “0”, “7”, “4”, “1”, “2”, “4”, “7”, “0”, “0”, “9”, “9”, “9”, “9”, “9”, “9”, “0”, “0”, “4”, “5”, “6”, “7”, “6”, “5”, “0”, “0”, “0”, “0”, “0”, “0”, “0”, “0”, “0”

If k= 10 then the program should return (5+3+0+2) = 10

If k= 7 then the program should return 0

If k= 23 then the program should return 0

If k= 21 then the program should return (2+2+5+8) = 17

**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** (8 points)

Write the KaprekarRoutine subroutine, which receives in input a 32-bit unsigned number *a*, ranging between 1000 and 9999. The subroutines implements the following tasks:

* it computes a new number *b* by sorting the digits of *a* by descending order
* it computes a new number *c* by sorting the digits of *a* by ascending order
* it returns *d* = *b* - *c*

Example.

a = 3075

The subroutine computes *b* = 7530 and *c* = 0357.

The returning value is *d* = 7530 – 357 = 7173.

You can obtain the digits of a number (from the least significant to the most significant) by repeatedly dividing by 10 and taking the remainder. For example:

3075 / 10 = 307 with remainder 5

307 / 10 = 30 with remainder 7

30 / 10 = 3 with remainder 0

3 / 10 = 0 with remainder 3

After sorting the digits, you can build the new numbers *b* and *c* by repeated multiplication by 10. In the previous example, the digits in descending order are 7, 5, 3, 0. You can build *b* as follows:

* first iteration: *b* = 7
* second iteration: *b* = 7 \* 10 + 5 = 75
* third iteration: *b* = 75 \* 10 + 3 = 753
* fourth iteration: *b* = 753 \* 10 + 0 = 7530

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. The assembly subroutine must comply with the ARM Architecture Procedure Call Standard (AAPCS) standard (in terms of parameter passing, returned value, callee-saved registers).
3. Click on the following links to open web pages with the ARM instruction set

https://developer.arm.com/documentation/dui0473/m/preface

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

**Question 5** (5 points)

Implement the handler for the supervisor call 50.

The handler repeatedly calls the KaprekarRoutine subroutine developed in the previous exercise, passing the value stored in r0. The loop ends when the KaprekarRoutine subroutine returns 6174. The handler sets r6 equal to the number of iterations (i.e., the number of times the subroutine was called).

Examples

If the supervisor call is called with r0 = 3075, the subsequent values of r0 are: 7173 -> 6354 -> 3087 -> 8352 -> 6174 and the handler returns 5.

It is proved that, for any four-digit numbers in base 10 (with at least two distinct digits), the Kaprekar's routine always yields the number 6174 after at most seven iterations. The value 6174 is known as Kaprekar's constant.

Note: within the handler, you can get the 16-bit operating code of the supervisor call by accessing the stack at position SP + 24 (you can assume that the calling program was using the main stack). The immediate value of the supervisor call is stored in the least significant byte.