**Question #1** (4 points)

Let’s consider the Thread Level Parallelism (TLP) seen during lectures.

You are requested to describe:

1. What is a thread;
2. What is the difference between ILP and TLP;
3. The different typologies of multithreading;
4. Their advantages and disadvantages, adding a short discussion about the best option.

Write your answer here.

Thread is a unit program can execute some instructions in parallel. Each core may have some threads to execute some instructions simultaneously.

ILP (Instruction Level Parallel) it depends on the pipeline and determines how many instructions can completely be executed in a unit time, and it depends on the architecture of the microprocessor.

The advantages of the threads are that some programs can be executed at the same time. Additionally, it can use GPU to help CPU to compute intensive tasks such as animation render, graphic tasks, scientific operations, and Machine Learning computations. But we should consider data dependency. If programs have data dependency each thread should wait till data is provided, therefore they can use stall.

**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 < 20; i++) {

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

; }

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

|  |  |  |
| --- | --- | --- |
| .data | Comments | Clock cycles |
| v1: .double “20 values” |  |  |
| v2: .double “20 values” |  |  |
| v3: .double “20 values” |  |  |
| v4: .double “20 values” |  |  |
| v5: .double “21 values” |  |  |
| v6: .double “20 values” |  |  |
|  |  |  |
| .text |  |  |
| main: daddui r1,r0,0 | r1 ← pointer |  |
| daddui r2,r0,20 | r2 ← 20 |  |
| loop: l.d f1,v1(r1) | f1 ← v1[i] |  |
| l.d f2,v2(r1) | f2 ← v2[i] |  |
| l.d f3,v3(r1) | f3 ← v3[i] |  |
| l.d f4,v4(r1) | f4 ← v4[i] |  |
| daddui r3,r1,8 | r3 ← r1 + 8 |  |
| l.d f5,v5(r3) | f5 ← v5[i+1] |  |
| mul.d f6,f1,f2 | f6 ← v1[i] \* v2[i] |  |
| mul.d f6,f6,f3 | f6 ← f6 \* v3[i] |  |
| add.d f6,f6,f4 | f6 ← f6 + v4[i] |  |
| sub.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 | F | D | E | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 5 |
| daddui r2,r0,20 |  | F | D | E | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |
| loop: l.d f1,v1(r1) |  |  | F | D | E | M | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 2 |
| l.d f2,v2(r1) |  |  |  | F | D |  | E | M | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 2 |
| l.d f3,v3(r1) |  |  |  |  | F |  | D |  | E | M | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 2 |
| l.d f4,v4(r1) |  |  |  |  |  |  | F |  | D |  | E | M | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 2 |
| daddui r3,r1,8 |  |  |  |  |  |  |  |  | F |  | D | E |  | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |
| l.d f5,v5(r3) |  |  |  |  |  |  |  |  |  |  | F | D |  | E | M | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 2 |
| mul.d f6,f1,f2 |  |  |  |  |  |  |  |  |  |  |  | F |  | D | E | E | E | E | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 3 |
| mul.d f6,f6,f3 |  |  |  |  |  |  |  |  |  |  |  |  |  | F | D |  |  |  | E | E | E | E | M | W |  |  |  |  |  |  |  |  |  |  |  | 4 |
| add.d f6,f6,f4 |  |  |  |  |  |  |  |  |  |  |  |  |  |  | F |  |  |  | D |  |  |  | E | E | M | W |  |  |  |  |  |  |  |  |  | 2 |
| sub.d f6,f6,f5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | F |  |  |  | D |  | E | E | M | W |  |  |  |  |  |  |  | 2 |
| s.d f6,v6(r2) |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | F |  | D |  | E | M | M | W |  |  |  |  |  | 2 |
| daddui r1,r1,8 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | F |  | D | E |  | M | W |  |  |  |  | 1 |
| daddi r2,r2,-1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | F | D |  | E | M | W |  |  |  | 1 |
| bnez r2,loop |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | F |  |  | D | E | M | W |  | 2 |
| halt |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | F |  |  |  |  | 1 |

**6+20\*29=586**

**Question 3** (6 points)

An 8 x 8 matrix XMATR 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 XFIELD. Write an 8086 program which, having received an index k of XFIELD, computes a value in XRES DB ? as follows:

* If XFIELD[k] is a border cell (i.e. if XFIELD[k]=”0”), then the program returns the value 0 in XRES
* Otherwise, the program returns in XRES the sum of the corresponding values of the cells which are its immediate diagonal (main and secondary, i.e. from left to right and from right to left) neighbors in XMATR; (the value of the cell should not be added).

a c

\*

d b

Please observe/comply with the following

* It is mandatory to cut the matrix by rows.
* In your solution, please provide the declaration of XFIELD and the code, together with a short description of the algorithm used and significant comments to the code and instructions.
* It is guaranteed that XMATR 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 XFIELD 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 XMATR

“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”

XFIELD = “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 (0+2+0+4) = 6

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 (4+9+7+9) = 29

**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 algoritm196 subroutine, which receives in input a 32-bit unsigned number *M*. If *M* (in the base-10 representation) is palindromic, the function returns 0. A palindromic number is a number (such as 16461) that remains the same when its digits are reversed.

If *M* is not palindromic, the function builds a new number *N* by reversing the digits of *M*. Then, the function returns *M* + *N*.

Examples

*M* = 126621. The function returns 0

*M* = 12661 -> *N* = 16621. The function returns 12661 + 16621 = 29282

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:

12661 / 10 = 1266 with remainder 1

1266 / 10 = 126 with remainder 6

126 / 10 = 12 with remainder 6

12 / 10 = 2 with remainder 2

1 / 10 = 0 with remainder 1

The loop ends when the result of the division is zero.

Since you do not know how many digits the number has, it is suggested to save each digit in the stack. Then, you can access the stack to check if the number is palindromic. If the number is not palindromic, you can build *N* by repeatedly multiplying the temporary value of *N* by 10 and adding the next digit (starting from the least significant one). In the previous example:

* first iteration: N = 0 \* 10 + 1 = 1
* second iteration: N = 1 \* 10 + 6 = 16
* third iteration: N = 16 \* 10 + 6 = 166
* fourth iteration: N = 166 \* 10 + 2 = 1662
* fifth iteration: N = 1662 \* 10 + 1 = 16621

You can assume that the computation never generates an overflow.

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 100.

The handler repeatedly calls the algoritm196 subroutine developed in the previous exercise, passing the value stored in r0. The loop ends when either one of the following conditions occur:

* the algoritm196 subroutine returns 0: it means that a palindromic number was reached. In this case, the handler sets r5 equal to 1 and ends.
* the algoritm196 subroutine has been called 10 times without returning 0. In this case, the initial value stored in r0 (when the handler was called) is a candidate to be a *Lychrel number*. A Lychrel number is a natural number that cannot form a palindrome through the iterative process of repeatedly reversing its digits and adding the resulting numbers. This process is called the 196-algorithm, because 196 is supposed to be the lowest Lychrel number. In this case, the handler sets r5 equal to 2 and ends.

Examples

If the supervisor call is called with r0 = 1879, the subsequent values of r0 are: 11660 -> 18271 -> 35552 -> 61105 -> 111221 -> 233332 -> 0 and the handler returns 1 (189 is not a Lychrel number).

If the supervisor call is called with r0 = 196, the subsequent values of r0 are: 887 -> 1675 -> 7436 -> 13783 -> 52514 -> 94039 -> 187088 -> 1067869 -> 10755470 -> 18211171 and the handler returns 2 (196 is a Lychrel number).

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.