**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.

1. It is a pipeline processor with 5 stage with an instruction set and load store instruction it means just load and store can have access to memory.
2. CISC processor does not have pipeline and instruction set and memory load/store instruction but it can perform complex instruction but RISC processor has pipeline and instruction set and load store instruction and have more speed(it is faster)
3. It has instruction with same size.
4. They have different pipeline stages.

-I got 1.5 for question one and 4 for question 2

-I did not get full grade of 8086, because I used DIV instruction and I had to use shift right

-I could not handle question 5 during exam

-I got 24 for this course

**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 | 5 |
| daddui r2,r0,30 | r2 ← 30 | 1 |
| loop: l.d f1,v1(r1) | f1 ← v1[i] | 1 |
| l.d f2,v2(r1) | f2 ← v2[i] | 1 |
| div.d f5,f1,f2 | f5 ← v1[i] / v2[i] | 7 |
| l.d f3,v3(r1) | f3 ← v3[i] | 0 |
| l.d f4,v4(r1) | f4 ← v4[i] | 0 |
| div.d f6,f3,f4 | f6 ← v3[i] / v4[i] | 6 |
| sub.d f7,f7,f2 | f7 ← –v2[i] | 0 |
| add.d f7,f7,f4 | f7 ← –v2[i] + v4[i] | 0 |
| mul.d f5,f5,f6 | f5 ← f5 \* f6 | 3 |
| add.d f5,f5,f7 | f5 ← f5 + f7 | 2 |
| s.d f5,v5(r2) | v5[i] ← f5 | 1 |
| daddi r2,r2,-1 | r2 ← r2 – 1 | 1 |
| daddui r1,r1,8 | r1 ← r1 + 8 | 1 |
| bnez r2,loop |  | 2 |
| halt |  | 1 |
| Total: |  | 26\*30+6 = 786 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| main: daddui r1,r0,0 | F | D | E | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 5 |
| daddui r2,r0,30 |  | F | D | E | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |
| loop: l.d f1,v1(r1) |  |  | F | D | E | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |
| l.d f2,v2(r1) |  |  |  | F | D | E | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |
| div.d f5,f1,f2 |  |  |  |  | F | D |  | E | E | E | E | E | E | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 7 |
| l.d f3,v3(r1) |  |  |  |  |  | F |  | D | E | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 0 |
| l.d f4,v4(r1) |  |  |  |  |  |  |  | F | D | E | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 0 |
| div.d f6,f3,f4 |  |  |  |  |  |  |  |  | F | D |  |  |  | E | E | E | E | E | E | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 6 |
| sub.d f7,f7,f2 |  |  |  |  |  |  |  |  |  | F |  |  |  | D | E | E | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 0 |
| add.d f7,f7,f4 |  |  |  |  |  |  |  |  |  |  |  |  |  | F | D |  | E | E | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 0 |
| mul.d f5,f5,f6 |  |  |  |  |  |  |  |  |  |  |  |  |  |  | F |  | D |  |  | E | E | E | M | W |  |  |  |  |  |  |  |  |  |  |  |  | 3 |
| add.d f5,f5,f7 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | F |  |  | D |  |  | E | E | M | W |  |  |  |  |  |  |  |  |  |  | 2 |
| s.d f5,v5(r2) |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | F |  |  | D | E |  | M | W |  |  |  |  |  |  |  |  |  | 1 |
| daddi r2,r2,-1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | F | D |  | E | M | W |  |  |  |  |  |  |  |  | 1 |
| daddui r1,r1,8 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | F |  | D | E | M | W |  |  |  |  |  |  |  | 1 |
| bnez r2,loop |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | F |  | D | E | M | W |  |  |  |  |  | 2 |
| halt |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | F | D | E | M | W |  |  |  |  | 1 |

**Question 3** (6 points)

A 7 x 7 matrix of bytes stores letters A-Z in upper case, with 4 and only 4 instances of the letter “A”. The matrix is cut by rows and implemented by the array FIELD. Write an 8086 assembly program which computes and stores to the array FIELD the positions of the four “A” letters and writes them to the array POSITION. The program should also write in the array DIAG a 1 if the corresponding “A” letter is on the main diagonal of the matrix (i.e. where i=j), i.e. from top left to bottom right or 0 if it is not. In other words:

POSITION (index of FIELD of first letter “A”) (index of FIELD of second letter “A”) (index of FIELD of third letter “A”) (index of FIELD of fourth letter “A”)

DIAG (1 if first letter “A” has i=j, 0 otherwise) (1 if second letter “A” has i=j, 0 otherwise) (1 if third letter “A” has i=j, 0 otherwise) (1 if fourth letter “A” has i=j, 0 otherwise)

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 the matrix only stores letter ascii characters and that there are only 4 instances of the letter “A”.
* 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 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 same column.

Example:

Matrix

C D **A** F K K J

B **A** B D H G R

O O P U Y R E

W W W W F R Y

T T T T T T T

D **A** E H T U I

R E R T S W **A**

FIELD = C, D, **A**, F, K, K, J, B, **A**, B, D, H, G, R, O, O, P, U, Y, R, E, W, W, W, W, F, R, Y, T, T, T, T, T, T, T, D, **A**, E, H, T, U, I, R, E, R, T, S, W, **A**

POSITION = 2 8 36 48

DIAG = 0 1 0 1

**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)

Given two areas of memory, the first one containing byte constants and the second one being uninitialized, write the copyData subroutine in ARM assembly language, which copies the content of the first area of memory to the second one. The subroutine receives in input:

* the address of the first area of memory
* the address of the second area of memory
* the number of elements declared in the first area of memory

The procedure does not return any value.

The size of the second area of memory is higher than or equal to the size of the first area.

Example:

AREA constants, DATA, READONLY

inputData DCB 3, -14, 15, -92, 65, 35, -89

AREA variables, DATA, READWRITE

outputData space 12

Then, write the insertionSort subroutine, which receives in input:

* the address of an area of memory (READWRITE)
* the number of elements in the area (7 in the previous example)

The procedure does not return any value. It sorts the elements, rewriting the area of memory, by means of the insertion sort algorithm. The pseudocode of the insertion sort is the following (A is the array to sort):

1. i ← 1
2. while i < length(A)
3. x ← A[i]
4. j ← i - 1
5. while j >= 0 and A[j] > x
6. A[j+1] ← A[j]
7. j ← j - 1
8. end while
9. A[j+1] ← x
10. i ← i + 1
11. end while

In the example above, if the subroutine receives the address of outputData, at the end the area contains the values -92, -89, -14, 3, 15, 35, 65.

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

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

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

**Question 5** (5 points)

Extend the project developed in the previous question as follows.

Initialize timer 1 to count from 0 to 0xFF. When the timer counter reaches 0xFF, it is reset and it starts counting again; no interrupt is generated.

Declare an uninitialized array of char. The size of the array is MAX\_VALUES, which is a positive constant that you can define as you prefer (e.g., MAX\_VALUES = 20).

When button INT0 is pressed, you have to:

* Read the value of the timer counter of timer 1
* Assign the value to the first available element (i.e., not yet initialized) of the array of char. At the first pressure of the button, the value of the timer counter will be saved at position 0 of the array; at the second pressure it will be saved at position 1, etc. If all elements of the array have already been initialized, the value of the timer counter is not used.
* Alternatively switch on led 6 and led 7 to signal the correct read of the timer counter. At the first pressure of the button, led 6 is switched on and led 7 is switched off. At the second pressure, led 6 it switched off and led 7 is switched on. At the third pressure of the button, led 6 is switched on and led 7 is switched off, etc.

When button KEY1 is pressed, you have to:

* Switch off led 6 and 7
* Call the insertionSort subroutine, passing the address of the array of char and the number of initialized elements. The subroutine will sort the elements of the array, considering them as signed byte values.
* When the subroutine ends, switch on led 11.

**Notes about the leds.** The pins of leds 4-11 are P2.7 – P2.0. The function LED\_init (included in the provided template) initializes the pins as GPIO Port 2.0 (LPC\_GPIO2). You have to switch on the required leds by means of the following accessible registers:

* FIODIR: Fast GPIO Port Direction control register. This register individually controls the direction of each port pin.
* FIOMASK: Fast Mask register for port. Writes, sets, clears, and reads to port (done via writes to FIOPIN, FIOSET, and FIOCLR, and reads of FIOPIN) alter or return only the bits enabled by zeros in this register.
* FIOPIN: Fast Port Pin value register using FIOMASK. The current state of digital port pins can be read from this register. The value read is masked by ANDing with inverted FIOMASK. Writing to this register places corresponding values in all bits enabled by zeros in FIOMASK.
* FIOSET: Fast Port Output Set register using FIOMASK. This register controls the state of output pins. Writing 1s produces highs at the corresponding port pins. Writing 0s has no effect. Reading this register returns the current contents of the port output register. Only bits enabled by 0 in FIOMASK can be altered.
* FIOCLR: Fast Port Output Clear register using FIOMASK. This register controls the state of output pins. Writing 1s produces lows at the corresponding port pins. Writing 0s has no effect. Only bits enabled by 0 in FIOMASK can be altered.