**Question 1** (4 points)

Let’s consider the MIPS64 processors seen during lectures.

You are requested to:

1. Describe what pipelined and unpipelined processors are;
2. Describe what is the Programmer’s Model of a generic processor;
3. Give the definition of Throughput and CPI;
4. Describe the techniques for managing branches in a pipeline.

Write your answer here.

1-Pipelined processors are allows the instructions to be executed in parallel and so that the instruction doesn’t have to wait previous instruction to be started. In addition, due to pipelined property the processor achieves better cycle per instruction.

2-The companies doesn’t want to include their all properties that can be known from other companies and copy their processors so, they are only allowing required part for the programmers that can use to implement their task

3-Throughput is a processor task that can implement per unit of time. The CYCLE PER INSTRUCTION is the amount of time that required to complete that specific instruction.

4-The branch prediction can be implemented in two ways statically or dynamically

In static branch prediction the compiler is incharge of implementation by analizing the code and make assumption whether the branches are more likely to be taken or not taken the examples can be;

-Branch always taken -Branch always not taken

-Prediction based on direction -Profile based prediction

The dynamic branch prediction is implemented by processor at the run time by restoring the Branh Prediction Unit.

The static approach is making the processor less complex but achieves lower performance. The Dynamic prediction is achieve higher performance with reasonable cost.

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

; }

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

|  |  |  |
| --- | --- | --- |
| .data | Comments | Clock cycles |
| v1: .double “20 values” |  |  |
| v2: .double “20 values” |  |  |
| v3: .double “20 values” |  |  |
| v4: .double “20 values” |  |  |
| v5: .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] |  |
| mul.d f5,f1,f2 | f5 ← v1[i] \* v2[i] |  |
| add.d f7,f7,f2 | f7 ← v2[i] |  |
| l.d f3,v3(r1) | f3 ← v3[i] |  |
| l.d f4,v4(r1) | f4 ← v4[i] |  |
| mul.d f6,f3,f4 | f6 ← v3[i] \* v4[i] |  |
| sub.d f7,f7,f4 | f7 ← v2[i] - v4[i] |  |
| div.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: |  | 27\*20+6=546 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| main: daddui r1,r0,0 | F | D | E | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |
| daddui r2,r0,20 |  | 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 |
| mul.d f5,f1,f2 |  |  |  |  | F | D |  | E | E | E | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 4 |
| add.d f7,f7,f2 |  |  |  |  |  | F |  | D | E | E |  | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |
| l.d f3,v3(r1) |  |  |  |  |  |  |  | F | D | E |  |  | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |
| l.d f4,v4(r1) |  |  |  |  |  |  |  |  | F | D |  |  | E | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |
| mul.d f6,f3,f4 |  |  |  |  |  |  |  |  |  | F |  |  | D |  | E | E | E | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 4 |
| sub.d f7,f7,f4 |  |  |  |  |  |  |  |  |  |  |  |  | F |  | D | E | E |  | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |
| div.d f5,f5,f6 |  |  |  |  |  |  |  |  |  |  |  |  |  |  | F | D |  |  | E | E | E | E | E | E | M | W |  |  |  |  |  |  |  |  |  | 6 |
| 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 |  |  |  | 1 |
| halt |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | F |  |  |  | W |  |  | 1 |

**Question 3** (6 points)

A 9 x 9 matrix of bytes stores letters A-Z in upper case, with 4 and only 4 instances of the letter “A” either in lower or upper case. The matrix is cut by rows and implemented by the array XFIELD. Write an 8086 assembly program which computes and stores to the array XFIELD the positions of the four “A” letters and writes them to the array XPOSITION. The program should also write in the array XDIAG a 1 if the corresponding “A” letter is on the secondary diagonal of the matrix (i.e. where i+j=8, i.e. from top right to bottom left) or 0 if it is not. In other words:

XPOSITION (index of XFIELD of first letter “A”) (index of XFIELD of second letter “A”) (index of XFIELD of third letter “A”) (index of XFIELD of fourth letter “A”)

XDIAG (1 if first letter “A” has i+j=8, 0 otherwise) (1 if second letter “A” has i+j=8, 0 otherwise) (1 if third letter “A” has i+j=8, 0 otherwise) (1 if fourth letter “A” has i+j=8, 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 XFIELD 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 D F K K J Z **A**

B **A** B D H G R K K

O O P U Y R E V R

W W W W F R Y R S

T T T T T T T O P

Z X C V B N M N B

D S E H T U I U X

R **A** R T S W **A** K U

Q W E R T Y U I P

XFIELD = C, D, D, F, K, K, J, Z, **A**, B, **A**, B, D, H, G, R, K, K, O, O, P, U, Y, R, E, V, R, W, W, W, W, F, R, Y, R, S, T, T, T, T, T, T, T, O, P, Z, X, C, V, B, N, M, N, B, D, S, E, H, T, U, I, U, X, R, **A**, R, T, S, W, **A**, K, U, Q, W, E, R, T, Y, U, I, P

XPOSITION = 8 10 64 69

XDIAG = 1 0 1 0

**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 bubbleSort 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 bubble sort algorithm. The pseudocode of the bubble sort is the following (A is the array to sort):

1. n ← length(A)
2. swapped ← TRUE
3. while swapped and n > 1
4. swapped ← false
5. for i = 1 to n - 1
6. if A[i - 1] > A[i] then
7. swap(A[i - 1], A[i])
8. swapped ← true
9. n ← n - 1

The loop at step 4 is repeated n – 1 times, i.e, i ranges from 1 to n – 1 included.

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 0 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\_LEN, which is a positive constant that you can define as you prefer (e.g., MAX\_LEN = 15).

When button INT0 is pressed, you have to:

* Read the value of the timer counter of timer 0
* 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 4 and led 5 to signal the correct read of the timer counter. At the first pressure of the button, led 4 is switched on and led 5 is switched off. At the second pressure, led 4 it switched off and led 5 is switched on. At the third pressure of the button, led 4 is switched on and led 5 is switched off, etc.

When button KEY1 is pressed, you have to:

* Switch off led 8 and 9
* Call the bubbleSort 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 10.

**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.