**COMP1411 (Group 2011) Introduction to Computer Systems**

Take-home exam Time: 12:30 ~ 14:30, 30-April-2022 (Saturday)

**Question Paper**

**Instructions:**

* You MUST type your answers into the provided answer book.
* DO NOT type your answers into this question paper.
* Remember to check the detailed instructions in the provided answer book.
* You MUST answer the questions by yourself only.
* You are NOT allowed to discuss questions or answers with other people.
* There are 7 questions in total, do not miss questions.

**Question 1. [15 marks]**

Consider the following C-language statement, where **a** and **b** are 64-bit unsigned integer variables.

**b = a \* 52;**

The above C statement is compiled into the following X86-64 assembly instructions. Note that the original value of variable **a** is stored in register **%rdi**, and the target value of variable **b** is stored in register **%rbx**. Assume **no overflow** occurs during the execution.

movq %rdi, %rax

movq %rax, %rdx

movq $ , %rcx /\* input an immediate number \*/

%rcx, %rax /\* input an instruction \*/

$1, %rcx /\* input an instruction \*/

%rcx, %rdx /\* input an instruction \*/

subq %rdx, %rax

shrq $ , %rcx /\* input an immediate number \*/

%rcx, %rdx /\* input an instruction \*/

subq %rdx, %rax

movq %rax, %rbx

**Fill in the six blanks in the above assembly code.**

Requirements:

* For each blank, you MUST follow the instructions specified in /\* … \*/.
* You are NOT allowed to introduce new registers other than %rax, %rbx, %rcx, %rdx.
* You are NOT allowed to use instructions other than movq, salq, shrq, subq.
* You do NOT need to provide steps for this question; please only fill in the blanks.

Functionalities of the instructions:

* movq a, b: moving a 64-bit data from register “a” to register “b”
* salq a, b: left shift the number in register “b” with “a” bits, the number in register “b” is 64-bit, and the operand “a” is an immediate number
* shrq a, b: unsigned right shift the number of register “b” with “a” bits, the number is register “b” is 64-bit, and the operand “a” is an immediate number
* subq a, b: subtract the value in register “a” from the value in register “b”, the values are 64-bit

**(Question 2 is on the next page)**

**Question 2. [10 marks]**

Assume that a system uses 8-bit floating-point representation based on the IEEE floating-point format, in which:

* There is 1 sign bit.
* There are k = 3 exponent bits.
* There are n = 4 fraction bits.

**sign**

**exponent**

**fraction**

We create an 8-bit number from the last two digits of your student ID (ignoring the last letter D). For example, assume your student ID is “12345678D”, then the last two digits are 78. Writing 7 into a 4-bit binary is 0111; writing 8 into a 4-bit binary is 1000. Then the 8-bit number we want to create is “01111000”.

**Compute the decimal value of the 8-bit IEEE floating-point number** created with the above method.

Requirements:

* You MUST explicitly show your steps.
* You MUST write the decimal number with the exact value.
* Do NOT write the number in the exponent form. For example, 1.25 x 23 is NOT allowed.

**(Question 3 is on the next page)**

**Question 3. [20 marks]**

Consider the execution of the following program (written in the C language).

**// all headers needed are included**

**int main()**

**{**

**int i = 1;**

**while (i < 4){**

**if (fork() != 0){**

**i++;**

**}**

**i++;**

**printf("%d ", i);**

**waitpid(-1, NULL, 0);**

**}**

**return;**

**}**

**3(a) Draw the process graph of the program. [10 marks]**

In a process graph, each invocation of any function, including main(), fork(), printf(), and waitpid(), is represented by a vertex. For each vertex, please write the function name below the vertex, and for printf() write the output character above the vertex. Each edge must be directed, with the direction representing the happen-before relationship.

Assumptions:

* Each time fork() is invoked, a child process can always be successfully created.
* printf() will always immediately print the content on your screen.
* “i++” does not appear in the process graph.

**3(b) How many numbers are there in each output? How many feasible outputs can be produced by this program? Why? [10 marks]**

Requirements:

* Identical outputs are deemed as one feasible output.
* Show how you compute the number of feasible outputs step-by-step. You do not have to list all the final possible outputs, as long as you can clearly explain the program behavior (how the outputs are generated in each part of the program) so that your number of feasible outputs is computed.

**(Question 4 is on the next page)**

**Question 4. [12 marks]**

Consider a virtual memory system with paging. A program is partitioned into several virtual pages, with the first virtual page numbered 0. The hardware main memory is also partitioned into several physical pages, with the first physical page numbered 0.

Assume the page size is 2KB (B = bytes), and each physical or virtual address is represented by a 16-bit binary number (or equivalently, a 4-digit hexadecimal number).

**Suppose two virtual addresses of the program are accessed sequentially**. The two virtual addresses are obtained from your student ID (ignoring the last letter D). For example, assume your student ID is “12345678D”, the first virtual address will be 0x1234, and the second virtual address will be 0x5678. Note that both 0x1234 and 0x5678 are hexadecimal numbers.

When each address is accessed, the corresponding virtual page should be first loaded into some physical page in the main memory. The mapping from a virtual page (of the program) to the physical page (of the main memory) is as follows: given a virtual page number X, the corresponding physical page number Y can be computed by: Y = X + 5.

**Write out the corresponding physical addresses of the above two virtual addresses.**

Requirements:

* You MUST provide steps to show how you compute each physical address from the corresponding virtual address.
* Write the final physical addresses in hexadecimal forms.

**(Question 5 is on the next page)**

**Question 5. [10 marks]**

Suppose a combinational logic is implemented by 8 serially connected steps named from A to H. The whole computation logic can be viewed as an instruction.

The time spent on each step comes from your student ID (ignoring the last letter D). For example, assume your student ID is “12345678D”, Then the time spent on each step is as follows:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| A | B | C | D | E | F | G | H |
| 1ms | 2ms | 3ms | 4ms | 5ms | 6ms | 7ms | 8ms |

Note that: 1 second = 1000ms.

**Make the computation logic a 3-stage pipeline design that has the maximal throughput.** Note that a register shall be inserted after each stage to separate their combinational logic, and the time spent on each register is 2ms.

**Throughput** is defined as how many instructions can be executed on average in one second for a pipeline, and the unit of throughput is IPS, instructions per second.

**Latency** refers to the time duration starting from the first component and ending with the last register operation finished; the time unit for latency is ms.

**5(a) Answer how to partition the 8 steps into 3 stages for maximal throughput.[6 marks]**

**5(b) Compute the throughput and latency for your pipeline design, with steps. [4 marks]**

Requirements:

* If there are multiple pipeline designs with the same maximal throughput, you only need to provide one of them.
* You MUST explicitly show your steps.
* For throughput value, only leave the integer part (with rounding to the nearest whole number). For example, if the value is 13.4, please give 13; if the value is 15.5, please give 16.

**(Question 6 is on the next page)**

**Question 6. [20 marks]**

We have a sequence of 4 assembly instructions (in the Y86-64 instruction set), and the corresponding machine code is as follows. The machine has a little-endian byte ordering.

**5030020000000000000030F11E00000000000000504213000000000000006067**

The execution behavior of the sequence of instructions is as follows:

* An instruction MUST sequentially go through 6 stages: Fetch (F), Decode (D), Execute (E), Memory (M), Write-back (W), and PC-update (P).
* The instruction will spend some time in each stage. The time spent in any stage depends on whether the instruction has operation in this stage. The time consumption is given in the following table.
* Only one instruction can execute at any stage. For example, if instruction 1 is currently executing in the E stage and instruction 2 wants to enter its E stage to execute, instruction 2 must wait for instruction 1 to finish execution in the E stage, and then instruction 2 can start its E stage.
* An instruction will start execution immediately when the F stage is free (i.e., the previous instruction has finished execution in its F stage).
* There is no extra stalling for any stage of any instruction, other than stage competition.

We make the following assumptions:

* The executions in the Fetch stage and in the Memory stage do NOT conflict, even though they all access memory.
* The register file can be accessed by multiple instructions in parallel.
* No instructions or data are cached.
* The program starts at time slot 1.

Time consumption in each stage:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Stages | F | D | E | M | W | P |
| Has operation | 3 | 2 | 2 | 4 | 2 | 2 |
| No operation | 1 | 1 | 1 | 1 | 1 | 1 |

**6(a) Write out the 4 instructions from the given machine code. [8 marks]**

**6(b) Compute** **the starting and end time of each instruction, and the time spent in each stage for each instruction, by filling the following table.** **[12 marks]**

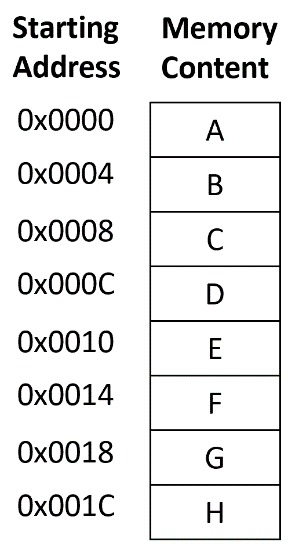
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Instruction** | **Start**  **Time** | **End**  **Time** | **F** | **D** | **E** | **M** | **W** | **P** |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |

For example, if the first instruction is “rrmovq %rax, %rbx”, the numbers in the table should be as follows. Note that this is only a demonstrative example, not part of the answer.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Instruction** | **Start**  **Time** | **End**  **Time** | **F** | **D** | **E** | **M** | **W** | **P** |
| rrmovq %rax, %rbx | **1** | **12** | **3** | **2** | **2** | **1** | **2** | **2** |
| **……** | **13** | **…** | **…** | **…** | **…** | **…** | **…** | **…** |

**(Question 7 is on the next page)**

**Question 7. [13 marks]**

We have **8 4-byte integer variables**, A, B, C, D, E, F, G, H, located continuously in the main memory, as shown in the figure on the right.

The main memory is partitioned into blocks with **each block having a size of 8 bytes**. The block starting from address 0x0000 is block 1, the next adjacent block is block 2, so on and so force. Each time a variable is accessed, the block it belongs to is loaded.

**The system has a cache that can hold 3 blocks**. When the program accesses a variable, the CPU first searches if the block containing the variable is in the cache or not. If the block is in the cache, we say there is a cache hit, and the CPU gets the variable from the cache; otherwise, we say there is a cache miss, and the CPU will fetch the block containing the variable from the main memory and at the same time put the block in the cache.

At the beginning of the program execution, **the cache is empty**.

Assume that **Least Recently Used (LRU)** is applied when replacing blocks.

The program is as follows, in which acc(X) means to access the variable X. We assume that all variables of the program, other than A, B, C, D, E, F, G, H, are all located in the registers and do NOT cause memory or cache access.

**// all headers needed are included**

**int main()**

**{**

**int i = 0;**

**acc(A);**

**while (i < 4){**

**if ((i % 2) ==0)**

**acc(B);**

**else{**

**acc(C);**

**acc(E);**

**}**

**acc(H);**

**i++;**

**}**

**acc(D);**

**return 0;**

**}**

**7(a) How many variable accesses are generated by this program? [1 mark]**

**7(b) Draw the cache state transitions as a result of the program execution. [12 marks]**

Requirements:

* Write the accessed variable above the arrow.
* Write out whether the access to the corresponding block is a cache hit (H) or miss (M) below the arrow.
* Fill in the cache states with block numbers.

An example of accessing variable A is given below: the vertical boxes represent a cache state with some blocks in the cache; an arrow represents the behavior of accessing a block, changing the cache state from the left one to the right one.

