CDT204 – Computer Architecture

Date: Nov. 2nd 2015

Time: 8:10 – 13:30

Help: Small calculator is allowed in the exam

The exam has 32 points. The grades will be awarded as follows:

- 3 : 17 Points
- 4 : 22 Points
- 5 : 27 Points

Important Notes:

- Give as full an answer as possible to obtain full marks. All calculations, approximations, assumptions and justifications must be reported for full credit unless stated otherwise. Please use figures and examples to clarify.
- If you do not understand a question clearly, you are allowed to call the teacher and ask.
- Write the question and part number on each page clearly.
- Answer each question on a separate page.
- Bonus questions will give you extra points (they raise the total over 32) ***if*** you can solve them.

May the force be with you.

Task 1 – General (6p)

Please answer the question **on this paper** and include it in the folder that you give back to the exam supervisors.

True or False? Mark the correct choice.

a. MIPS instructions have fixed length. — O [✓ True] O False

b. An instruction takes less time to execute on a pipelined processor than a non-pipelined processor if everything else is the same between the two processors. — O True O [✓ False]

c. On the MIPS computer, a jump (J) instruction can use a wider range of immediate addresses than a branch instruction because more bits are allocated for the address in the jump instruction than the branch instruction. — O [✓ True] O False

d. The binary representation of the number 77 is different in Two's Complement format than the unsigned format. — O True O [✓ False]

e. A multi-cycle implementation of MIPS requires that a single memory be used for both instructions and data. — O True O [✓ False]

f. The choice of programming language affects the performance of the final code. — O [✓ True] O False

g. The string representation of the number 999,999,999 requires more memory than its binary form (assume that you are coding in C language). — O [✓ True] O False

h. The speed of the memory system affects the designer's decision on the size of the cache block. Therefore, for a memory with higher bandwidth it is better to use a larger cache block. — O [✓ True] O False

i. GPUs normally have smaller caches compared to CPUs, because they rely on heavy threading and data-parallelism. — O [✓ True] O False

j. Thread management is a complex and difficult issue for parallel GPU programmers and that is why GPU programming is not popular. — O True O [✓ False]

k. An Instruction Set can be implemented in different designs and architectures which in turn will affect its performance. — O [✓ True] O False

l. Assuming a cache size of 4KB and 16 bytes cache block size, a fully associative cache will be harder to implement than a 256-way set-associative cahe. — O True O [✓ False]

m. Skynet is real, because Moore's Law. — O True O True O [✓ ! False]

Task 2 – Performance (6p)

Salvatore and Jean-Philippe work as CPU designers for Putayto Chips Inc. Their new processor core design has the following CPIs for different classes of instructions:

Instruction Type	**CPI**
Arithmetic	1
Load / Store	12
Branch	5

Their design can be organized as a single core or multiple core architecture. A benchmark program that Salvatore and Jean-Philippe use, requires to execute the following number of instructions on the single core design:

Instruction Type	**Instruction Count**
Arithmetic	2.56×10^9
Load / Store	1.28×10^9
Branch	256×10^6

The design runs on a clock frequency of 2 GHz for each processor on both single and multiple core setups. In order to test the multicore version, the program in parallelized and the required changes are made in the code to make it possible to run over multiple cores. In this program, the number of arithmetic and load/store instructions per processor is divided by 0.7 x *p* (where *p* is the number of processors) but the number of branch instructions per processor remains the same.

***Hint**: for example on a dual core setup (p=2), each core will have* $\frac{2.56\times10^9}{0.7\times2} = \frac{2.56\times10^9}{1.4} = 1.83\times10^9$ *arithmetic instructions to execute.*

a. Find the total execution time for this program on 1, 2, 4, 8 processors and the relative speedup of the 2, 4, 8 processor result relative to the single processor result.
b. If the CPI of the arithmetic instructions was doubled, what would the impact be on the execution time of the program on 1, 2, 4, 8 processors?
c. To what should the CPI of load/store instructions be reduced in order for a single processor to match the performance of four processors using the original CPI values?

Solution part A

First we calculate the number of instructions for each of the multicore setups.

	Arithmetic Instructions Count	*Load/Store Instructions Count*
1 Core	2.56×10^9	1.28×10^9
2 Cores	$(2.56 \times 10^9)/(0.7\text{x}2)=1.83 \times 10^9$	$(1.28 \times 10^9)/(0.7\text{x}2)=0.91 \times 10^9$
4 Cores	$(2.56 \times 10^9)/(0.7\text{x}4)=0.91 \times 10^9$	$(1.28 \times 10^9)/(0.7\text{x}4)=0.46 \times 10^9$
8 Core	$(2.56 \times 10^9)/(0.7\text{x}8)=0.46 \times 10^9$	$(1.28 \times 10^9)/(0.7\text{x}8)=0.23 \times 10^9$

Calculating the execution times are straightforward, the execution time will be

$$execution\ time = \frac{instruction\ count \times CPI}{clock\ frequency}$$

Since we have different classes of instructions, we need to calculate them separately and add them together, so we will have:

$$\boldsymbol{Execution\ time\ (1\ Core)} = \frac{(2.56 \times 10^9 \times 1) + (1.28 \times 10^9 \times 12) + (256 \times 10^6 \times 5)}{2 \times 10^9}$$

$$= \frac{2.56 \times 10^9 + 15.36 \times 10^9 + 1.28 \times 10^9}{2 \times 10^9} = \frac{(2.56 + 15.36 + 1.28) \times 10^9}{2 \times 10^9} = \mathbf{9.6}\ \boldsymbol{sec}$$

$$\boldsymbol{Execution\ time\ (2\ Cores)} = \frac{(1.83 \times 10^9 \times 1) + (0.91 \times 10^9 \times 12) + (256 \times 10^6 \times 5)}{2 \times 10^9}$$

$$= \frac{1.83 \times 10^9 + 10.92 \times 10^9 + 1.28 \times 10^9}{2 \times 10^9} = \frac{(1.83 + 10.92 + 1.28) \times 10^9}{2 \times 10^9} = \mathbf{7.015}\ \boldsymbol{sec}$$

$$\boldsymbol{Execution\ time\ (4\ Cores)} = \frac{(0.91 \times 10^9 \times 1) + (0.46 \times 10^9 \times 12) + (256 \times 10^6 \times 5)}{2 \times 10^9}$$

$$= \frac{0.91 \times 10^9 + 5.52 \times 10^9 + 1.28 \times 10^9}{2 \times 10^9} = \frac{(0.91 + 5.52 + 1.28) \times 10^9}{2 \times 10^9} = \mathbf{3.855}\ \boldsymbol{sec}$$

$$\boldsymbol{Execution\ time\ (8\ Cores)} = \frac{(0.46 \times 10^9 \times 1) + (0.23 \times 10^9 \times 12) + (256 \times 10^6 \times 5)}{2 \times 10^9}$$

$$= \frac{0.46 \times 10^9 + 2.76 \times 10^9 + 1.28 \times 10^9}{2 \times 10^9} = \frac{(0.46 + 2.76 + 1.28) \times 10^9}{2 \times 10^9} = \mathbf{2.25}\ \boldsymbol{sec}$$

So the speedups will be

$$Speedup\ (2\ Cores) = \frac{Execution\ time\ (1\ Core)}{Execution\ time\ (2\ Cores)} = \frac{9.6}{7.015} = 1.37$$

$$Speedup\ (4\ Cores) = \frac{Execution\ time\ (1\ Core)}{Execution\ time\ (4\ Cores)} = \frac{9.6}{3.855} = 2.49$$

$$Speedup\ (8\ Cores) = \frac{Execution\ time\ (1\ Core)}{Execution\ time\ (8\ Cores)} = \frac{9.6}{2.25} = 4.26$$

Solution part B

The formulas are the same, we will have

$$Execution\ time\ (1\ Core) = \frac{(2.56 \times 10^9 \times \mathbf{2}) + (1.28 \times 10^9 \times 12) + (256 \times 10^6 \times 5)}{2 \times 10^9} = \frac{5.12 + 15.36 + 1.28}{2} = 10.88 sec$$

$$Execution\ time\ (2\ Cores) = \frac{(1.83 \times 10^9 \times \mathbf{2}) + (0.91 \times 10^9 \times 12) + (256 \times 10^6 \times 5)}{2 \times 10^9} = \frac{3.66 + 10.92 + 1.28}{2} = 7.93 sec$$

$$Execution\ time\ (4\ Cores) = \frac{(0.91 \times 10^9 \times \mathbf{2}) + (0.46 \times 10^9 \times 12) + (256 \times 10^6 \times 5)}{2 \times 10^9} = \frac{1.82 + 5.52 + 1.28}{2} = 4.31 sec$$

$$Execution\ time\ (8\ Cores) = \frac{(0.46 \times 10^9 \times \mathbf{2}) + (0.23 \times 10^9 \times 12) + (256 \times 10^6 \times 5)}{2 \times 10^9} = \frac{0.92 + 2.76 + 1.28}{2} = 2.48 sec$$

Solution part C

From part A we have the execution time of 4 core design which is 3.855. We want to achieve the same result with 1 CPU so we will have:

$$3.855 = \frac{(2.56 \times 10^9 \times 1) + (1.28 \times 10^9 \times \boldsymbol{X}) + (256 \times 10^6 \times 5)}{2 \times 10^9}$$

Where ***X*** is the new CPI of load/store instructions. We solve for X:

$$3.855 = \frac{(2.56 \times 10^9 \times 1) + (1.28 \times 10^9 \times \boldsymbol{X}) + (256 \times 10^6 \times 5)}{2 \times 10^9} \Rightarrow$$
$$3.855 = \frac{(2.56 + 1.28X + 1.28) \times 10^9}{2 \times 10^9} \Longrightarrow 3.855 = \frac{1.28 \times (2 + X + 1)}{2} \Longrightarrow$$
$$7.71 = 1.28 \times (3 + X) \Longrightarrow 6 = 3 + X \ \Longrightarrow \boldsymbol{X} = \mathbf{3}$$

So we should reduce the CPI for load/store instructions from 12 to 3.

Task 3 – MIPS Assembly (3p)

Consider the following MIPS program

```
.global __start

.data
x:  .word 256

.text
__start:
         addi $s0, $zero, 0
         addi $s1, $zero, 1
         lw   $s2, x
loop:
         beq  $s2, $s1, end
         div  $s2, $s2, 2
         addi $s0, $s0, 1
         j    loop
end:
         ...
```

a. What will the value of $s0 register be when the program reaches point “end”? (The ‘DIV’ instruction performs integer division). Motivate your answer.

This code calculates the $\log_2 n$ (it divides the number by 2 until reaching zero and counts the number of steps). Therefore the result will be 8. Of course you can also follow the code line by line and reach the same result.

b. Write the equivalent C code for the above MIPS code.

```
int x = 256;

main(){
    int a=0;
    while(x>1){
        x = x / 2;
        a = a + 1;
    }
    ...
}
```

Task 4 – Pipelining (5p)

Assume the following five steps MIPS data path and their latencies

Pipeline stage	Description	Latency(ps)
IF	Instruction Fetch – Get the next instruction from memory	250
ID	Instruction Decode – Decode the op-code and read the register file	350
EX	Execute – Perform arithmetic operations / compare commands / calculate memory addresses	150
MEM	Memory – Read from or write to memory	300
WB	Write Back – Write back the result to the register file	200

Also assume that the instructions executed by the processor are broken down as follows

ALU	BEQ	LW	SW
45%	20%	20%	15%

a. What is the clock cycle time and frequency in a pipelined and non-pipelined processor?

In a non-pipelined processor all the steps should be performed during one cycle, so the cycle time will be equal to summation of the latencies for all the step which will be

250+350+150+300+200 = 1250ps

In a pipelined system, the clock cycle can be a short as the longest step which in this case is the ID step. Therefore a pipelined processor in this case will have a cycle time of 350ps.

The frequencies will be as follows

$$Frq.Non-pipelined = \frac{1}{1250 \times 10^{-12}} = \frac{10^{12}}{1.25 \times 10^{3}} = 0.8 \times 10^{9} = 0.8GHz = 800Mhz$$

$$Frq.pipelined = \frac{1}{350 \times 10^{-12}} = \frac{10^{12}}{.35 \times 10^{3}} = 2.87 \times 10^{9} = 2.87GHz = 2870Mhz$$

b. What is the total latency of an LW instruction in a pipelined and non-pipelined processor?

In a non-pipelined version, the latency for all the instructions is the same and equal to clock cycle, so in this case the latency of LW for a non-pipelined system will be 1250ps. In a pipelined system the latency of LW will be the summation of all the steps that the instruction will require (which in the case of LW is all the steps) but all the steps also have the same latency as the longest one, therefore:

350ps x 5 (steps) = 1750ps

c. If we can split one stage of the pipelined data path into two new stages, each with half the latency of the original stage, which stage would you split and what are the new clock cycle and frequency of the processor?

If we want to split one of the stages into two, then the best option is to break down the longest stage. In this case the ID stage will be broken into ID1 and ID2 each with 175ps latency. In such a case the cycle time and frequency for the non-pipelined version stays the same, but for the pipelined version, now the longest step is MEM stage which takes 300ps, therefore

$$Frq.pipelined = \frac{1}{300 \times 10^{-12}} = \frac{10^{12}}{.3 \times 10^{3}} = 3.33 \times 10^{9} = 3.33GHz = 3333Mhz$$

d. Assuming there are no stalls or hazards, what is the utilization of the data memory?

The instructions that use the data memory are LW and SW instructions. LW accounts for 20% of all the instructions and SW for 15% of them, so in total 35% of the instructions will require the data memory and memory utilization will be 35%.

e. Assuming there are no stalls or hazards, what is the utilization of the write-register port of the "Registers" unit?

The write-register port is used when an instruction wants to change the value of a register. This happens with ALU instructions (since they need to write the result back to a register) and LW instruction (since it needs to write the fetched data from memory to the register). Therefore the utilization will be 45+20=65%

Task 5 – Branch Prediction (5p + 2p BONUS)

Assume the following repeating pattern (for example in a loop) of branch outcomes: T, NT, T, T, NT (T: Taken, NT: Not Taken)

a. What is the accuracy of always-taken and always-not-taken predictors for this sequence of branch outcomes?

Always-taken: 3 correct out of 5 → 3/5*100=60%

Always-not-taken: 2 correct out of 5 → 2/5*100=40%

b. What is the accuracy of the two-bit predictor for the first 4 branches in this pattern, assuming that the predictor starts off in the bottom left state (predict not taken) of the following diagram?

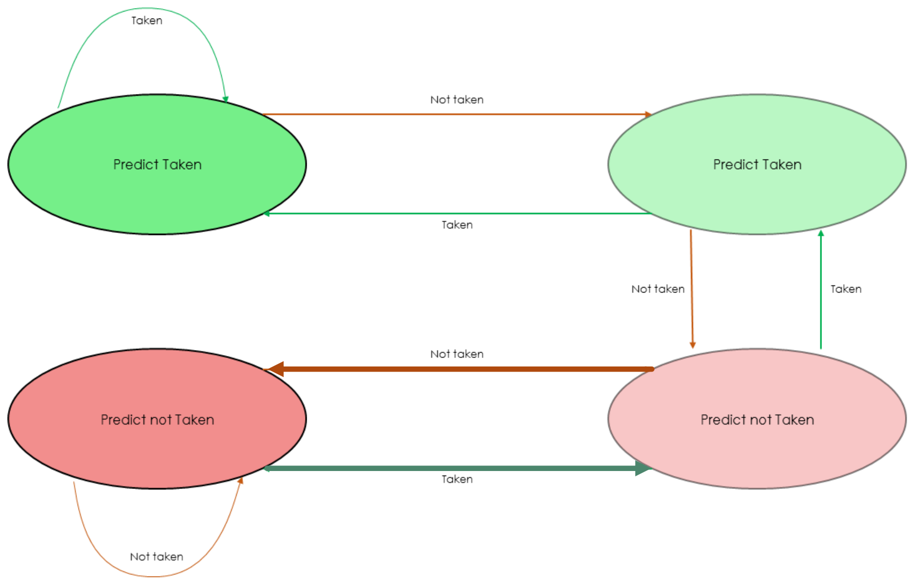
Step	Prediction	Actual Outcome	Notes
1	NT	T	The predictor changes to weakly not-taken state
2	NT	NT	The predictor moves back to strongly not-taken state
3	NT	T	The predictor changes to weakly not-taken state
4	NT	T	The predictor changes to weakly taken state
5	T	NT	The predictor changes to weakly not-taken state

So based on the above table, the predictor has only 1 correct prediction for its first 4 predictions. Therefore its accuracy is only 25%.

c. What is the accuracy of the two-bit predictor if this pattern is repeated forever?

If you follow the states while repeating the pattern, you will notice that the state gradually moves to Strongly Taken state and can never move from that state to non-taken states because there are no consecutive NT states. This means that the predictor will acts similar to an always-taken predictor in an infinite repetition situation and therefore the accuracy will be 60%.

d. *BONUS QUESTION (+2 points)*: Design a predictor that would achieve a perfect accuracy if this pattern is repeated forever.



Task 6 – Cache Memory (5p)

For a direct-mapped cache design with a 32-bit address, the following bits of the address are used to access the cache

Tag	Index	Offset
31-12 (10 bits)	11-5 (7 bits)	4-0 (5 bits)

a. What is the cache block size (in words)?

Block size = $2^{\#\text{offset bits}}$ = 2^5 = 32 bytes or 32/4=8 words

b. How many entries does the cache have?

#Entries = $2^{\#\text{index bits}}$ = 2^7 = 128

c. What is the cache size?

Cache size = Number of entries x Block size = 128 x 32 = 4096 Bytes = 4KB

Assume the following C program is run

```c
int arr[2048];

void main(){
        arr[4] = 20;                // 1
        arr[1029] = 8;              // 2
        arr[7] = 2008;              // 3
        arr[9] = arr[4];            // 4
        arr[10] = arr[9] + 7;       // 5

}
```

Suppose that the array is allocated by the compiler in a way that it starts at an address divisible by 4096 and that the array is the only thing that will occupy the space in the data cache. Also assume that an `int` takes up 4 bytes of space (which in turn means that the array will take up 8192 bytes).

d. Which of the memory accesses from the program will generate a cache miss and why if the cache is direct mapped?

Since we know that the array is allocated at an address divisible by the cache size, it can be inferred that it is possible to calculate which ache block will handle those memory accesses although that we do not know the exact base addresses. What we cannot know is that what tags will those addresses get (which is not important in this case). Let's call the base address ***b***, we will have:

1. Address: b + 16_{10} or b + 000000010000_2, which gives index 0. Miss since the block has not been used before.
2. Address: b + 4116_{10} or b + 1000000010100_2, which gives index 0 (The most significant bit that contains a 1 will be in the tag part). Miss since the block has not been used before. Notice that the block replaces the block that was read in (1))
3. Address: b + 28_{10} or b + 000000011100_2, which gives index 0. Miss since the block replaces (2).

4. Address 1: b + 36_{10} or b + 000000100100_2, which gives index 1.
   Address 2: the same ad-dress and index as (1).
   Miss for address 1 since the block has not been used before. Hit for address 2 since the block is read in (3).
5. Address 1: b + 4010 or b + 0000001010002, which gives index 1.
   Address 2: the same ad-dress and index as address 1 in (4).
   Hit for both references since both of the cache blocks that they are mapped to are read in (4).

Task 7 – Multiprocessing (3p)

Flynn's taxonomy in its classical form (represented in the table below) defines 4 different classes in computer architecture from a multiprocessing point of view.

	Single Instruction	**Multiple Instruction**
Single Data	SISD	MISD
Multiple Data	SIMD	MIMD

a. Explain each of the classes and give examples for each of them.

SISD: Single Instruction Single Data – This is the traditional way of processing. There is one data stream and one instruction stream. At each step the processor is busy on performing one instruction. Most of the older generation of CPUs work in this manner (i.e. Pentium 4)

MISD: Multiple Instruction Single Data – There are no products in this category to use as an example. But in theory such machines can perform a series of computations on a single data stream (i.e. in a pipelined fashion).

SIMD: Single Instruction Multiple Data – This is a more popular design, where one instruction can be performed on many data streams. A good example might be SSE and/or AVX instructions of x86 instruction set, where one instruction can be performed on a vector (array) of data. A GPU might also be considered as a SIMD processor, although it fits better to another category on extended version of Flynn's taxonomy.

MIMD: Multiple Instructions Multiple Data – These processors have several instruction streams (several instruction fetch units) and can perform different tasks in parallel with each other. Most of the processors today are designed with this concept in mind. A good example can be Intel Core i7.

b. Some researchers have proposed 2 newer classes called SPMD (Single Program Multiple Data) and MPMD (Multiple Programs Multiple Data). SPMD defines a design in which multiple processors execute the same program at independent points on different data at the same time. Can you give an example of hardware which has an architecture very similar to SPMD?

MIMD best fits the description of a GPU. GPUs have many cores that run the same program (Single Program) on different data (Multiple Data).

c. *BONUS QUESTION (+2 Points):* NIVIDIA classifies its architecture for Tesla and Fermi line of products as SIMT (Single Instruction Multiple Threads), compare this design with classical SIMD and/or Vector architecture (for example Intel AVX).