Computer Architecture: Spring 2001 – Test 1 Solutions

1. (12 pts) A measure of performance in a computer system is:

   EXECUTION TIME = CP x IC x CPI

a. DEFINE each term. CP = clock period, IC = instruction count, CPI: clocks per instruction.

b. For the 'IC' term, which of the following factors has the MOST effect in REDUCING this term (circle only one). ISA is instructuction set architecture; IMPLEMENTATION refers to implementation approach such as multi-cycle, pipelined, superscalar; technology refers to physical realization (i.e, 0.25u CMOS technology)

   ISA IMPLEMENTATION TECHNOLOGY

   The ISA determines if a program will take a few instructions or a lot of instructions to implement.

c. For the 'CP' term which of the following factors has the most effect in reducing this term? (circle only one)

   ISA IMPLEMENTATION **TECHNOLOGY**

   Technology has biggest impact on clock period - fast transistors means a smaller clock period.

d. For the 'CPI' term which of the following factors has the most effect in reducing this term? (circle only one).

   ISA **IMPLEMENTATION** TECHNOLOGY

   Implementation will effect CPI the most -- pipelined processor have a CPI of about 1.2, superscalar processors have a CPI < 1.0.

2. (8 pts) Assume that I have a feature on a processor that is used 50% of the time. If I speed this up by a factor 4X, what is the overall speedup of my processor?

Use Amdahl's law: Actual Speedup = 1/ [(1-f) + f/n]. F = 0.5, N = 4
Actual Speedup = 1/[(1-0.5) + 0.5/4] = 1.6

3. (12 pts) a. Explain the basic RISC design philosophy and how it differs from older processors designed using the CISC philosophy.
   RISC processors include only the most commonly used instructions and addressing modes-- this will make the control simpler which will speed up the clock period and reduce die area. More die area leaves room for more registers, so RISC processors tend to have more registers than older processors. RISC processors rely on a compiler using primitive instructions to emulate more complicated, but less used instructions found in older processors designed using a CISC style.

b. How are these philosophy differences demonstrated for instruction encoding on a RISC vs a CISC processor? *RISC processors have a FIXED WIDTH instruction encoding with fixed position encoding (i.e., all instructions 32 bits wide with register fields in fixed positions). This allowed for simpler control, especially for pipelined and superscalar implementations. CISC processors use a variable number of bytes to encode instructions. This was needed because of the complex instructions that were supported.*

c. How are these philosophy differences demonstrated for ALU instruction types on a RISC vs a CISC processor? *Obviously there are fewerALU instruction types on a RISC processor than a CISC processor (ie, RISC processors do not have special instructions for BCD arithmetic) . The most important change is that there are also fewer addressing modes allowed -- RISC ALU instructions can only use registers and immediate values. CISC ALU instructions are allowed to access memory.*

4. (8 pts) The MIPS architecture does not use traditional condition code flags for branches. Instead, it has only two branches (branch not equal, branch on equal). To do a more complicated branch (like branch on greater than), you have to use a SET instruction that will set a register to 0 or 1 depending on the SET type, then follow this with a branch instruction.

   a. WHY did the MIPS take this approach? Discuss some advantages of this approach.
   First, MIPS does include branch not equal, branch on equal which happen to be the most commonly used branches (make the common case fast!!!). Having the SET instruction store the "condition" in a general purpose register means that the compiler is free to move the SET and BRANCH instructions around, perhaps inserting instructions between them. This is important for an optimizing compiler, especially in superscalar implementations. Furthermore, by having a SET instruction, you only set a 'condition' code when you WANT TO, not as a default for every instruction. This preserves the condition code between multiple instructions.

   b. What is a disadvantage? *Using an entire 32-bit general purpose register to store only a '1' or '0' is somewhat of a waste. You end up dedicating a general purpose register for only a 1 bit flag. It also increases code size by requiring two instructions for a complex branch.*

5. (12 pts) a. What are REGISTER WINDOWS on the SPARC architecture? What is the benefit for having register windows? *Register windows are a set of overlapping register sets. A section of code can only view 32 registers at a time (8 global registers, 24 local). The benefit is that it allows the SPARC to use a lot of registers without having to increase instruction encoding size (5-bit fields are used in the instructions to specify the register number, all instructions are fixed width at 32-bits.). The OUT registers of one register set overlap with the IN registers of the next registers set.*

6. b. If Subroutine A wants to pass two parameters to Subroutine B on the SPARC, what registers will it use? *It will use OUT registers 0 and 1 (%o0, %o1).*

   c. Assume Subroutine B allocates a new register window. What two registers will it look at to receive its two parameters from Subroutine A?
   The parameters will be in the IN registers 0 and 1 (%i0, %i1).

7. (6 pts) The MIPS instruction "la r5, msg1" will load the 32 bit address represented by 'msg1' into register r5. This is really a psuedo instruction on the MIPS and takes two instructions. Write the two instructions that will accomplish this on the MIPS without using psuedo instructions.

   lui $r5, msg1(higher 16 bits)
   ori $r5, $r0, msg1(lower 16 bits)

8. (10 pts) CIRCLE the following items that you would find in an INSTRUCTION SET ARCHITECTURE definition:
The ISA specifies implementation independent features:

a. Listing of registers (*yes, implementation independent)*

b. Listing of pinout definition of the chip *(NO, specific to a particular implementation)*

c. Clock Speed rating *(NO, specific to a particular implementation)*

d. Cycle-by-cycle bus operation of instructions *(NO, specific to a particular implementation)*

e. Listing of instructions with addressing modes for each instruction (yes)

f. Listing of bytes codes (machine formats) for instructions (yes, each implementation must use the same instruction format for object code upward compatibility).

9. (5 pts)
The MIPS uses an instruction code format that has FIXED positions for register numbers (i.e., if an instruction has three operands, the register numbers are always in the same three 5-bit fields). What is the advantage of using fixed position operand encoding?
This allows the register values to be FETCHED before the instruction type is even identified (the instruction decoding can occur in parallel with register fetching). This improves execution speed.

10. (12 pts)
a. How does a register file differ from a normal memory (i.e. Static RAM use for cache)?
A register file is smaller, faster, and has multiple read, write ports. A typical static RAM memory only has one port on it, but can store more data (has more locations).

b. For a MIPS ALU instruction, how many READ ports and how many WRITE ports must the register file support? *Two read ports, one write port.*

b. Adding more read, write ports to register file is necessary to support SuperScalar processing (executing more than one instruction at a time). How does the size of the register file grow as I add more read and write ports?
The area of the register file grows as the SQUARE of the number of ports (area grows as ports^2). This is because each additional port adds wires in both the X and Y directions across each register file bit.

10. (15 pts) Write a MIPS program that has a "MAIN" entry point and a subroutine called "GETLEN". The main program will prompt the user to enter a string. The main program will pass the address of the string to a subroutine called *'strlen'* that will count the number of bytes in the string and return this value to main. The main program will print this value out to the user.

Enter String: *hello, I am typing string!*
#of Bytes: 26

Use the SPIM system calls for printing strings, getting the string from the user, and printing the number. Recall that the SPIM system call that gets the string from the user returns a NULL-terminated string (last byte in string is a zero byte = 0x00). The maximum size string the user can enter is 64 bytes. You may use psuedo instructions in your program.

You must use MIPS register naming conventions for passing arguments to and receiving arguments from subroutines! ***SEE LINK ON WWW PAGE TO PROBLEM SOLUTION.***