**Max Score = 15 points**

CS 250 2018 Spring Homework 05 SOLUTION & GRADING GUIDE

This assignment is due at 11:59:00 pm Thursday, February 22, 2018.

Upload your typewritten answer document in either PDF or Word format to Blackboard. Then, download from Blackboard to be sure that your upload was successful.

Your last upload that is not marked “LATE” by Blackboard is the upload that will be graded. There is no “grace” period for late uploads.

1. A program is stored in instruction memory starting at byte address 0x000000F0. The program contains 65 fixed-length 4-byte instructions.
   1. What is the byte address for the first byte of the last instruction in the program as stored in memory?  
      0x000001F0
   2. What is the byte address of the final byte of this program?  
      0x000001F3
   3. How many bytes of computer memory does this program occupy?  
      260
2. The 32-bit string0x10AF8007 is interpreted using the instruction format shown in Figure 6.2. What is a commented line of assembly language corresponding to this bit string?  
   00010 0001 0101 1111 000000000000111

regA unused dst offset  
LOAD dst reg, 7(regA) ; r15 🡨 Data\_Memory[7 + r1]

1. Using the instruction set architecture (ISA) format of Figures 6.1 and 6.2 in our textbook, what is the memory dump for the following code snippet?   
    Present the dump using a fixed width font; a framework is provided for you below. Each line of the dump will contain the following: (1) the hexadecimal starting address in a 32-bit byte-addressed memory, followed by (2) the memory contents, presented in binary, corresponding to one instruction, where any single bit of the instruction that can be either a 0 or a 1 is represented using the “don’t care” symbol, “X”, with (3) a single space separating each field in the instruction format.  
     
   Memory addr. Label Assembly instr. ; Comment  
   0x000000F0 L01: LOAD r1, 16(r4) ; r1 🡨 Data\_Memory[16 + r4]  
   0x000000F4 ADD r3, r2, r1 ; r3 🡨 r2 + r1  
   0x000000F8 STORE r6, 22(r4) ; Data\_Memory[22 + r4] 🡨 r6  
     
   Answer (fill in the rest of the answer):  
   Address opcode rA rB dst 15-bit offset

0x000000F0 00010 0100 X 0001 000000000010000

0x000000F4 00001 0010 0001 0011 XXXXXXXXXXXXXXX

0x000000F8 00011 0100 0110 XXXX 000000000010110

1. Consider textbook Figure 6.4. When executing a program (carrying out the Fetch-Execute cycle), the most commonly occurring case as determined by examining lots of actual program code has been found to be that the next instruction to fetch and execute is the instruction in memory that immediately follows the current instruction. Thus, for machines using a fixed-length 4-byte instruction representation and having byte-addressed memory  
    next\_instr\_ptr 🡨 current\_instr\_ptr + 4  
   is the formula to use by default to find the next instruction. Figure 6.4 implements this formula. In the figure, “32-bit pgm. ctr.” is the register that always outputs the bit string called current\_instr\_ptr. Pointers always are interpreted using the representation format of unsigned integer.
   1. What should be the representation and the corresponding bit string of the integer 4 appearing in Figure 6.4?

Next\_instr\_ptr = current\_instr\_ptr +4  
00000000000000000000000000000100

* 1. What is the destination register for the output of the adder in Figure 6.4?  
     It is the program counter
  2. What is the format that will be applied to the output bit string from the 32-bit adder in Figure 6.4?

Unsigned integer

New Address = previous address + 00000000000000000000000000000100

1. The circuit in Figure 6.4 can only compute the address for the most common case of the location of the next instruction to fetch. So, the machine must have both an instruction and the circuitry to handle less common cases for the location of the next instruction.  
    The JUMP instruction is the part of the ISA in Chapter 6 to allow the machine to be able to execute program having structures such as *if* statements and loops. JUMP computes a pointer to the next instruction independently of the circuit in Figure 6.4 by using the ALU adder circuit instead of the dedicated adder circuit shown in Figure 6.4.
   1. Using Figure 6.2 as a definition of the JUMP instruction, what are the names of the two fields of the two sources of the operands of the JUMP instruction?  
      regA and offset
   2. Categorize each JUMP instruction operand field as either an immediate value or a pointer. Also say what kind of circuit each pointer points to.  
      regA->immediate value

Offset->pointer

Offset points to multiplexer circuit

* 1. Referring to Figure 6.2, which JUMP operand(s) must use a multiplexer circuit to choose the physical location of the operand bit string?  
     offset
  2. Registers in the Register Unit of the example machine of Chapter 6 are all 32-bit in size. What are the sizes of the JUMP instruction operand bit strings as they are specified by the JUMP instruction?  
     32 bits
  3. The JUMP instruction computes the formula  
      next\_instr\_ptr 🡨 offset + contents[reg\_A]  
     For the hardware to perform this computation correctly both the offset value and the value stored in reg\_A must be the intended values at the time these bit strings are presented as inputs to the ALU adder circuit. The software development process encompasses the time from first designing (architecting) a program until the time that there is a machine program stored in instruction memory. If a program is written in a high-level language, what is the latest time in the software development process at which the value represented by the offset field must be known: at coding time, at compile time, or at run time?  
     at run time
  4. If writing an assembly language program, what is the latest time at which the value represented by the offset field of a JUMP instruction must be known?  
     at run time
  5. At what time, if any, during the software development process is contents[reg\_A] known?  
     at coding time
  6. By writing an assembly language code snippet, show how to use the MIPS processor instructions LOAD UPPER IMMEDIATE (mnemonic LUI) along with SHIFT RIGHT LOGICAL (mnemonic SRL) seen in Figure 5.9 along with the JUMP instruction from Figure 6.2 to implement the *do forever* looping action of the program of Figure 4.5.  
      A code snippet is a few lines of code, possibly taken out of a larger but unknown context.  
      A JUMP instruction would be the last instruction in the body of the *do forever* loop. The address that the JUMP instruction would compute would be the address of the first instruction of the *do forever* loop. This computation would be straightforward if only the values of both operands of a JUMP instruction were known at coding time, but they are in general not both known at that time. Thus, a little help from LUI and SRL is needed.  
      Write your assembly language code snippet based on the following assumptions. Shortly after this machine receives power, all registers are cleared (all bits set to zero), including the current\_instr\_prt register. Fetching instructions starts only after registers are cleared. Machine instructions are 4 bytes long. There is a 17-instruction assembly language preamble prior to the infinite loop in Figure 4.5 (a typical purpose for preamble code is to initialize some aspect of the automatic door system). The complete program for the automatic door starts at address 0x00000000 in memory. The instruction SRL (A)N shifts the bit string in the register named A by N bits to the right, destroying bits that shift past the register LSB and inputting zero bits to the register MSB as needed. The instruction LUI (A)N places the 16-bit unsigned integer N into the 16 most significant bits of the register named A. Register r1 may be used as you wish without any effect on the Figure 4.5 program assembly language instructions that are outside of your snippet. The offset value for the JUMP instruction is equal to zero. Constants in your snippet will be interpreted as decimal by default and as base 16 if preceded by 0x.  
      Write your code snippet using the instruction mnemonics. Provide a comment with each assembly language instruction to describe the operation of the instruction.
  7. Assume that both positive and negative integer values are desired for values in the offset field of the ISA of Figure 6.2. Figure 6.9 shows that the offset field bus connects to one of the two inputs buses of mux M2 and that the M2 output bus connects to the ALU. If the ALU has only one integer adder circuit, what integer representation format should be used for the offset field of this ISA?  
     00000000000000000000000000000100
  8. What is the range of the immediate values that can be provided using the offset field of the Figure 6.2 ISA?  
     0 to 16383
  9. Exactly what must be done to prepare the offset field before delivering its bits to the 32-bit adder in the ALU?  
     To have the instruction go through instruction decoder
  10. What is the integer representation of the ALU adder output when executing a JUMP instruction?  
      next\_instr\_ptr 🡨 offset + contents[reg\_A] while offset is a bit string of 0s and 1s of 15bits
  11. Figure 6.9 in the textbook shows multiplexer M1 having input buses for (1) the output of the dedicated “32-bit adder” which is the bus carrying the value default\_next\_instr\_ptr and for (2) a bus coming from the output of the ALU, if multiplexer M3 is set to select its input bus from the ALU rather than set to select its input bus from “data memory.” Mux M1 selects the bit string on one of its two input buses to pass along to the M1 output bus that connects to the “32-bit pgm. ctr.”  
       Complete the table below showing the address bit that each of the four instructions of the ISA of Figure 6.2 should cause to be delivered to the address bit input of M1 so that the Fetch-Execute cycle will always fetch the next instruction from the correct instruction memory address. Let M1 address 0 point to the default\_next\_instr\_ptr bus.

|  |  |
| --- | --- |
| ISA instruction mnemonic | Value of address bit to send to M1 address input |
|  |  |
|  |  |
|  |  |
| JUMP |  |

* 1. If register r0 holds the value 4 and the 2’s complement offset bit string represents -8, then executing the Chapter 6 ISA assembly instruction  
      JUMP -8(r0) ; next\_instr\_ptr 🡨 r0 + -8  
     will produce a negative integer result expressed in 2’s complement format. When this bit string appears on the output of the “32-bit pgm. ctr.” in Figure 6.4 and is delivered by bus to computer memory, it will be interpreted as a memory address, that is as a pointer, which always use the unsigned integer format: the hardware will be cast the bit string from 2’s complement to unsigned.  
     What address, in hexadecimal notation, will be received by the memory, and what part of a program’s address space is likely to include this address?