**Max Score = 15 points**

CS 250 2018 Spring Homework 07

This assignment is due at 11:59:00 pm Thursday, March 08, 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. Suppose you are designing a processor for a computer that will use a very slow memory technology when values are not stored in a register circuit. Will you choose a zero-address architecture? Why or why not?

No, I will not. Zero-address architecture fetches operands from memory and if memory is very slow, the speed is not ideal.

1. What is the greatest number of bits for an immediate operand of a fixed length ISA having a fixed size opcode?

15

1. Suppose that in addition to other hardware, the CPU used in a smartphone contains additional hardware to enable running software for three previous versions of the of the chip (i.e., three backward compatibility modes). What is the disadvantage from a smartphone user’s point of view?

It will increase the cost and also eat up more energy because there are more chips.

The following is information defining an ISA to use for answering Questions 4, 5, 6, and 7.

This is the ISA for a 32-bit computer, so the default integer size is 32 bits.

ISA machine code uses a 32-bit fixed length bit string in one of two instruction formats shown in the table. Fields of the instruction format are defined using [x – y] notation that means bit positions x through y, inclusive, are the bits of the field.

Machine instruction formats.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Instructions using three registers | Opcode [31 – 26] | Operand\_reg\_1 [25 – 21] | Operand\_ reg\_2 [20 – 16] | Result\_reg [15 – 11] | Unused [10 – 0] |
| Instructions using < 3 registers | Opcode [31 – 26] | Operand\_reg\_1 [25 – 21] | Result\_reg  [20 – 16] | Offset [15 – 0] | |

ISA instruction definition table.

|  |  |  |
| --- | --- | --- |
| Mnemonic | Opcode | Definition |
| ADD | 000001 | Result\_reg 🡨 Operand\_reg\_1 + Operand\_reg\_2 |
| LOAD | 000010 | Result\_reg 🡨 Data\_memory[Operand\_reg\_1 + Offset] |
| STORE | 000011 | Data\_memory[Operand\_reg\_1 + Offset] 🡨 Operand\_reg\_2 |
| JUMP | 000100 | Next\_instr\_ptr 🡨 Operand\_reg\_1 + Offset |
| BRR | 000101 | Next\_instr\_ptr 🡨 Default\_next\_instr\_ptr + Offset |
| BEQ | 000110 | If Operand\_reg\_1 == Operand\_reg\_2  then Next\_instr\_ptr 🡨 Default\_next\_instr\_ptr + Offset |
| BNE | 000111 | If Operand\_reg\_1 != Operand\_reg\_2  then Next\_instr\_ptr 🡨 Default\_next\_instr\_ptr + Offset |
| LOADHI | 001000 | Result\_reg[bits 31 – 16] 🡨 Offset |
| SRL | 001001 | Result\_reg 🡨 Logical-right-shift of Operand\_reg\_1 by Offset bit positions |
| ADDI | 001010 | Result\_reg 🡨 Operand\_reg\_1 + Offset |

1. How many registers can this ISA address?  
   3
2. What is the range of the 2’s complement Offset field in terms of powers of 2 and as integers?  
   -2^14 to 2^14-1
3. Using the ISA defined above, write assembly language to implement the code snippet  
    for (k=0; k<10, k++) {  
    a[0] = a[0] + a[k];  
    }  
    k = 5;  
     
   See Figure 9.3 in the text for a template framework for the assembly language.  
    **MAKE ADDITIONS IN BOLD FONT** **and blue color** to the code in the following table to create an assembly language program corresponding to the C code. Additions that are not in **BOLD FONT and blue color** will be ignored and not scored.  
    Some rows in the table already are filled in completely or partially. Do not modify the existing content in the table when you add information; your code must build on and be compatible with the contents of the already complete or partially complete rows.  
    Add labels only where needed. Make no assumptions about the bit strings stored in registers. Constant values are presented decimal (base ten) unless prefaced by 0x to indicate that the value is in hexadecimal (base 16). Use [x – y] notation to indicate a bit field where needed in your answer. You must include a correct, meaningful comment for each line of assembly if there is not a comment given in the table.

|  |  |  |
| --- | --- | --- |
| Label | Instruction | Comment |
|  | LOADHI r5, 0x0078 | r5[31 – 16] 🡨 0x0078; load upper half of r5 |
|  | SRL r5, 16 | r5 🡨 0x00000078; initialize lower half of r5 as desired |
|  | LOADHI r5, 0x0C11 | r5 🡨 0x0C110078, address of a[0] now constructed in r5 |
|  | ADDI **r6, r5, 0** | Copy contents of r5 into r6 |
|  | **LOADHI r1, 0** | r1[31 – 16] 🡨 0 |
|  | **SRL r1, 16** | r1 🡨 0 ; initial value of k placed in register r1 |
|  | **LOADHI r2, 10** | **R2[31-16] 🡨 0x000A**; starting to place 10 in r2 |
|  | SRL r2, 16 | r2 🡨 10 ; upper bound of k constructed in register r2 |
|  | LOAD r3, 0(r5) | **Load r3 using a[0]** |
| Startfor: | r1, r2, Endfor | If r1  **==**  r2 then no more iterations are needed |
|  | **LOAD r4 0(r)** | Load a[k] into r4 |
|  | **ADD r3, r3, r4** | a[0] 🡨 a[0] + a[k] |
|  | **JUMP r4, r6** | Make r6 point to a[k] for the next loop iteration |
|  | **ADDI r1, r1, 1** | perform k++ |
|  | **BRR startfor** | End of the loop body |
|  | LOADHI r1, 5 | r1[31 – 16] 🡨 5 |
|  | **SRL r1 16** | **R1 🡨 0; initialize value of k placed in register r1** |

1. Assemble the following code into machine code and write your answer in the provided table. Unused fields are to be filled entirely with 1 bits. The address of label L1 is 0x00400000.  
    L1: LOAD r9, 12(r0)  
    ADD r11, r9, r10  
    ADDI r0, r0, 4  
    BRR L1

|  |  |
| --- | --- |
| Machine instruction binary string | In 0x notation |
| **000010 00000 01001 0000000000001100** | 0x0809000C |
| **000001 01001 01010 01011 11111111111** | 0x052A5FFF |
| **001010 00000 00000 0000000000000100** | 0x28000004 |
| **000101 11111 11111 1111111111110000** | 0x17FFFFF0 |