Michael Lankford

Homework 3

2/14/2023

1.

* The Instruction Set Architecture is the interface between the hardware and lowest level software that encompasses all the information necessary to write a machine language program.
* Machine code is language that the computer understands and is based on 0s and 1s. An assembly program is written in assembly language, which is a symbolic version of machine instructions. A high-level program is a portable language that is composed of words and arithmetic notation that can be translated by a compiler into assembly language.
* There are several instruction classes in MIPS:
  + Arithmetic: Each instruction performs one operation and must always have 3 variables / Ex: add a, b, c
  + Branch: Each instruction compares 2 values and transfers the comparison to a new address / Is conditional / Ex: bne reg1, reg2, L1
  + Jump: Each instruction will jump to the label of the target instruction / Is unconditional / Ex: L1: sub ecx, 1

jal L1

* + Data Transfer: transfer data between registers and memory / Load and store are examples of this because they are used to load data and store data in memory / Ex: lw $t0, 32($s30)
  + Logical: Used to pack and unpack bits into words / Ex: sll $t2, $s0, 4
* There are 3 operand types in MIPS:
  + Register operands: Used for arithmetic instructions in the MIPS architecture / Some register operand examples are $t0, $t1, $t2, etc.
  + Memory operands: Used to transfer data between memory and registers in the MIPS architecture / Is essentially a large, 1-D array with the address acting as the index to that array / Some memory operand examples are: load and load word(l and lw)
  + Constant operands: Useful for speeding up operations because you don’t have to load the constant from memory / An example of a constant operand is: addi
* There are 3 instruction formats in MIPS:
  + R-Format: also called register format / Has 6 instruction fields: op, rs, rt, rd, shamt, and funct / Ex: add $t0, $s1, $s2
  + I-Format: also called immediate format / Used by the immediate and data transfer instructions / Has 4 instruction fields: op, rs, rt, and constant or address / Ex: lw $a3, 52($s3)
  + J-Format: also called jump format / loads an immediate value into the PC register / Has 2 instruction fields: op and target address / Ex: L1: sub ecx, 1

jal L1

* There are 5 addressing modes in MIPS:
  + Register addressing mode: Simplest addressing mode because it doesn’t have to deal with memory access / Is a form of direct addressing / Ex: add $t0, $t1, $t2
  + Immediate addressing mode: Contains one operand that is a constant within the instruction itself / Better when needing a constant because it doesn’t need to have extra memory access to fetch the constant / Ex: j nextCh
  + Base addressing mode: Also known as indirect register addressing / The operand is at the memory location whose address is the register / Ex: lb $t0, ($t2)
  + PC-relative addressing mode: Used for conditional branches / The address is the sum of the program counter and a constant in the instruction / Ex: beqz $t0, strEnd
  + Pseudo direct addressing mode: The memory address is embedded in the instructions and the address must be a multiple of 4 / Ex: j label
* Big endian is the order where the most-significant bit is stored first, at the lowest storage address. Little endian is the order where the least significant bit is stored first.
* Sign extension replicates the most significant bit of a signed binary number while preserving its value. Zero extension adds 0s in front of a unsigned binary number. An example of sign extension is slt $t0, $s0, $s1. An example of zero extension is sltu $t0, $s0, $s1.

2. exercise 2.4

|  |  |  |
| --- | --- | --- |
| sll | $t0, $s0, 2 | # $t0 = f \*4 |
| add | $t0, $s6, $t0 | # $t0 = &A[f] |
| sll | $t1, $s1, 2 | # $t1 = g \* 4 |
| add | $t1, $s7, $t1 | # $t1 = &B[g] |
| lw | $s0, 0($t0) | # f = A[f] |
| addi | $t2, $t0, 4 | # $t2 = &A[f + 1] |
| lw | $t0, 0($t2) | # $t0 = A[f + 1] |
| add | $t0, $t0, $s0 | # $t0 = A[f + 1] + A[f] |
| lw | $t0, 0($t1) | # $t0 = B[g] |

3. exercise 2.5

|  |  |  |  |
| --- | --- | --- | --- |
| Big-endian | | Little-endian | |
| Address | Data | Address | Data |
| X + 3 | ab | X + 3 | 12 |
| X + 2 | cd | X + 2 | ef |
| X + 1 | ef | X +1 | cd |
| X | 12 | X | ab |

4. exercise 2.7

|  |  |  |
| --- | --- | --- |
| sll | $t0, $s3, 2 | # $t0 = i \*4 |
| add | $t0, $s6, $t0 | # $t0 = &A[i] |
| lw | $t0, 0($t0) | # $t0 = A[i] |
| sll | $t1, $s4, 2 | # $t1 = j \* 4 |
| add | $t1, $s6, $t1 | # $t1 = &A[j] |
| lw | $t1, 0($t1) | # $t1 = A[j] |
| add | $t0, $t0, $t1 | # $t0 = &(A[i] + A[j]) |
| sw | $t0, 32($s7) | # B[8] = A[i] + A[j] |