भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

Computer Architecture

Assignment -2 - CS2323

Kartik Srinivas - ES20BTECH11015
October 11, 2022

Index

1 Section 1 **2**

2 Section 2 **3**

3 Section 3 **4**

1 Section 1

The instruction format along with the binary and then the Hexadecimal is being reported

1. Instruction Format - I type
   Immediate(7 + 4) - Binary for -4 (2's complement) = 1111 1111 1100
   rs1 - x22 - 22 - 10110
   funct3 - 0x0 - 000
   Opcode - 0010011
   Net Instruction = **0xffcb0793**

2. Instruction format R - type funct7 - rs2 -rs1 - funct3 -rd -opcode
   rs2 - 9 - 01001
   rs1 - 01000
   funct3 -0x4
   rd - 10111
   - opcode - 011011
   Net Instruction - **0x00944b3**

3. Instruction Format - B type - imm - rs2- rs1- funct3- imm -opcode
   Immediate - 240 = 0 0000 1111 0000
   rs2 - 00111
   rs1 - 1 - 00001
   funct3 - 0x0
   opcode - 1100011
   Net Instruction - **0x0e138863**

4. Instruction Format S - type - imm - rs2 - rs1 - funct3 - imm - opcode
   Immediate - 24 - 0 000 000 11000
   rs1 - 9 - 01001 - rs1 is the one to which the immediate is added (source of data)
   rs2 - 00110
   funct3 - 0x3
   opcode - 0100011
   Net Instruction - **0x0064bc23**

5. Instruction type - J - type - imm - rd - opcode
   Immediate = -1080 - 2'c complement - 0000 0000 1011 1100 1000
   rd - 8 - 01000
   opcode - 1101111
   Net Instruction - **0x83dfd46f**

---

2 Section 2

The load immediate instruction can be considered as a "pseudo instruction".

1. li t0 -1 = addi x5 x0 -1
   Clearly these both have the same meaning, we load into the register t0 = x5 after adding a '-1' to x0 = 0

2. li t0 0xFFFFFFFF
   Note that the hexadecimal 0xFFFF FFFF is equivalent to -1 (it is the 2's complement of 1, so both the instructions are the same)

3. li t0 223
   223 is only 8 bits long, so it within the 12 bit immediate that addi can handle, so the instruction addi x5 x0 223, remains valid

4. li t0 1234
   Note that 1234 is 11 bits long which is still within the valid length of 12 bits for immediates , so addi x5 x0 1234 will still be correct ($1234 < 2^{11} - 1$)

5. li t0 0x123456000
   Now there has been a change, there are 9 hexadecimal digits which equals 36 normal bits, **too large** for an immediate, so it needs to be added by breaking it into pieces. The first step done by the simulator is to load 0x00092 into bits 31 : 12 using lui, then it adds the number -1493 = 0xfffffffa2b
   on addition , the number becomes 0x00000000091a2b, this number on shifitng left by 13 will then become 0x123456000.

---

3 Section 3

The first instruction will load bits 12 through 31 inside the register x3, using the value that is stored , the next instruction will load half-word (in an unsigned manner)
Half word = 16 bits = 4 hexadecimal digits. The lower 4 Hexadecimal digits from byte 0 and byte 1 will be used for storage within the least significant bits of the register x3.
x3 = 0x000000000000a5a5
Now if instead we load in a signed manner, the CPU will first check the MSB of the 16 bit number being loaded. This a5a5 has the sign bit of **0xa = 1010** .Therefore there is an **extension** of 1's while loading (which is equivalent to the
x3 = 0x$ffffffffffffa5a5$
Now loading from an offset of 2 will load **Byte 3 and Byte 4** into the register along with thw usual sign extension
x3 = 0x$ffffffffffffa5_{byte-4}a5_{byte-3}$
Now we load the entire double from Byte-0 onwards, we will get the first double word inside the register.
x3 = 0x39933939a55aa5a5
Now we load from **Byte 5 onwards**. Since the memory is contiguous, it will start reading into the **next double word** The number stored is then a mix of the higher bytes from the first doubble word and the lower of the second one.
x3 = 0x$39933939_{second}39933939_{first}$
Next instruction will just load the eight'th byte(7 have been offset)
x3 = 0x0000000000000039
Now even if I use **lb** , it won't make a difference because **0x3 = 0011** and the sign extension will be just of 0's.
x3 = 0x0000000000000039
Now , 7'th Byte actually is **0x93 = 1001 0011**, thsi would imply a sign extension when loaded
x3 = 0xffffffffffff93
For code please see [Sri13]

References

[Sri13] Kartik Srinivas. Cs2323 - computer architecture. https://github.com/kartiksrinivas007/CS2323-CArch.git, 2013.