CS 350 – COMPUTER ORGANIZATION AND ASSEMBLY LANGUAGE PROGRAMMING
Week 5

Reading:

1. D. Patterson and J. Hennessy, Chapter- 4

Objectives:

- To understand load store architecture.
- Learn how to implement Load store instruction.

Concepts:

1. Load Store architecture
2. Load Store Instruction set
3. Advantages and disadvantages

Outline:

- Load Store Architecture
- Load Store Instruction set
    1. LB
    2. SW
    3. SB
    4. LW.

- Semantics of Instructions
- Examples how to Implement instructions
- Machine Code Representation.
- Advantage & Disadvantages.
- Memory Management

References:

1. **Fundamentals and principles of computer design by Joseph D. Dumas**.

What is Load Store Architecture

The alternative to a memory-register architecture is known as a Load-store architecture. Notable examples of the load-store philosophy include the Silicon Graphics MIPS and Sun SPARC processors. In a load-store architecture, only the data transfer instructions(typically named load for reading from memory and store for writing data to memory) are able to access variables in memory. All air thematic and logic instructions operate only on data in registers (or possibly immediate constants); they leave the results in registers as well. Thus, a typical computational instruction might be written in assembly language as ADD R1, R2, R3, while a data transfer might appear as LOAD[R4], R5. Combinations of the two, such as ADD R1, [R2], R3 are not allowed.

Load Store Instruction Set

- **lw** Loads a word from a location in memory to a register. Address in memory must be word-aligned.
- **lb** Loads a byte from a location in memory to a register. Sign extends this result in the register.
- **lbu** Loads a byte (unsigned) from a location in memory to a register. Zero extends the result in the register.
- **sw** Store a word from a register to a location in memory. Address in memory must be word-aligned.
- **sb** Store the least significant byte of a register to a location in memory.

Semantics

```
lw      $rt,    offset($rs)
lb      $rt,    offset($rs)
lbu     $rt,    offset($rs)
sw      $rt,    offset($rs)
sb      $rt,    offset($rs)
```

- Load byte instruction lb transfers one byte of data from main memory to a register.
  lb $t0, 20($a0) # $t0 = Memory[$a0 + 20]

- Store byte instruction sb transfers the lowest byte of data from a register into main memory.
  sb $t0, 20($a0) # Memory[$a0 + 20] = $t0

- load or store 32-bit quantities—a complete word instead of just a byte—with the lw and sw instructions

.

```
lw $t0, 20($a0) # $t0 = Memory[$a0 + 20]
sw $t0, 20($a0) # Memory[$a0 + 20] = $t0
```

Examples how to Implement Instructions

Consider the semantics of **lw**, and then explain the semantics of the other instructions.

$$Addr \leftarrow R[s] + (IR_{15})^{16} :: IR_{15-0}$$
$$R[t] \leftarrow M_4[\ Addr\ ]$$

The address is computed by adding the contents of register **s** to the sign-extended offset (which is an immediate value). This may result in an address that is not word-aligned (i.e., whose binary address does not end in 00). If that happens, a hardware exception occurs.

The four bytes located at memory address starting at addres, *Addr*, are copied to register **t**. The CPU fetches the four bytes based on the endianness of the CPU. Thus, the CPU arranges the bytes correctly within the register depending on whether the machine is big or little endian.

For **lb**, the address is computed the same way as **lw**, but the address does not have to be word aligned. The semantics of **lb** are shown below.

$$Addr \leftarrow R[s] + (IR_{15})^{16} :: IR_{15-0}$$
$$R[t] \leftarrow (M_1[\ Addr\ ]_7)^{24} :: M_1[\ Addr\ ]$$

Only one byte is loaded from memory. However, since this must be stored in a 32 bit register, and since the quantity is interpreted as 2C, the fetched byte is sign-extended to 32 bits. This is what $\mathbf{(M_1[\ Addr\ ]_7)^{24}}$ means (it's the sign bit copied 24 times).

lbu is just like **lb** except the byte is zero-extended in the register.

sw is similar to **lw** except the four byte quantity is copied from the register to memory.

$$Addr \leftarrow R[s] + (IR_{15})^{16} :: IR_{15-0}$$
$$M_4[\ Addr\ ] \leftarrow R[t]$$

When the contents of register **t** is written to memory, it is stored in the endianness of the CPU.

sb is similar to **sw**. The least significant byte of register **t** is copied to the address in memory.

```
Addr <- R[s] + (IR15)16::IR15-0
M1[ Addr ] <- R[t]7-0
```

Machine Code Representation

All of the load/store instructions are **I-type** since they use a 16-bit 2C immediate value as the offset.

Instruction	B_{31-26}	B_{25-21}	B_{20-16}	B_{15-0}
	opcode	**register s**	**register t**	**immediate**
lw $rt, <offset>$rs)	**100 011**	-	-	**offset**
lb $rt, <offset>$rs)	**100 000**	-	-	**offset**
lbu $rt, <offset>$rs)	**100 100**	-	-	**offset**
sw $rt, <offset>$rs)	**101 011**	-	-	**offset**
sw $rt, <offset>$rs)	**101 100**	-	-	**offset**

The dashes are 5-bit encoding of the register number in UB. For example, **$r7** is encoded as 00111. The offset is represented in 16-bit 2C.

Advantages and Disadvantages

- The advantages and disadvantages of a load store architecture are essentially the converse of t hose of a memory-register architecture.
- Assembly language programming requires a bit more effort
- Programs tend to require more machine instructions to perform the same task
- Operands must be in registers, more registers must be provided or performance will suffer.
- More registers are more difficult to manage
- More time-consuming to save and restore when that becomes necessary.
- Register addressing requires smaller bit fields (five bits are sufficient to choose one of 32 registers, while memory addresses are typically much longer)
- Instructions can be smaller (and perhaps more importantly, consistent in size).
- Control logic is less complex and can be implemented in hardwired fashion,
- Shorter CPU clock cycle.
- The arithmetic and logic instructions, which never access memory after they are fetched are simple to pipeline because data memory accesses are done independently of computations, they do not interfere with the pipelining as much they otherwise might.

Memory Management

In the load/store architecture only load and store instructions move data between the registers and memory. RISC machines as well as vector processors use this architecture which reduces the size of the instruction substantially. If we assume that memory addresses are 32 bit long, an instruction with all three operands in memory requires 104 bits whereas the register-based operands require only 23 bits .

Lab Practice

Assembly language programming Using Load Store Architecture based on 8085.

- Load byte instruction lb transfers one byte of data from main memory to a register.
- Store byte instruction sb transfers the lowest byte of data from a register into main memory
- Implement all the instruction Lw, Sw, Lb, Sb.