SYSTEM SOFTWARE

Module – I

Introduction: System software and machine architecture, traditional (CISC) machines, RISC machines.

Introduction

The subject introduces the design and implementation of system software. Software is set of instructions or programs written to carry out certain task on digital computers. It is classified into system software and application software. System softwareconsists of a variety of programs that support the operation of a computer. Application software focuses on an application or problem to be solved. System software consists of a variety of programs that support the operation of a computer. Examples for system software are Operating system, compiler, assembler, macro processor, loader or linker, debugger, text editor, database management systems (some of them) and, software engineering tools. These software's make it possible for the user to focus on an application or other problem to be solved, without needing to know the details of how themachine works internally.

System Software and Machine Architecture

One characteristic in which most system software differs from application software is machine dependency.

System software – support operation and use of computer. Application software - solution to a problem. Assembler translates mnemonic instructions into machine code. The instruction formats, addressing modes etc., are of direct concern in assembler design. Similarly, Compilers must generate machine language code, taking into account such hardware characteristics as the number and type of registers and the machine instructions available. Operating systems are directly concerned with the management of nearly all of the resources of a computing system.

There are aspects of system software that do not directly depend upon the type of computing system, general design and logic of an assembler, general design and logic of a compiler and, code optimization techniques, which are independent of target machines. Likewise, the process of linking together independently assembled subprograms does not usually depend on the computer being used.

The Simplified Instructional Computer (SIC)

Simplified Instructional Computer (SIC) is a hypothetical computer that includes the hardware features most often found on real machines. There are two versions of SIC, they are, standard model (SIC), and, extension version (SIC/XE) (extra equipment or extra expensive).

1. SIC standard Model
2. SIC/XE(extra equipment or expensive)

Object programs for SIC can be properly executed on SIC/XE which is known as upward compatibility.

SIC Machine Architecture/Components –

1. Memory –

- Memory is byte-addressable that is words are addressed by the location of their lowest-numbered byte.
- There are 2^15 bytes in computer memory (1 byte = 8 bits)
  3 consecutive byte = 1 word (24 bits = 1 word)
  - **2. Registers –**
  - There are 5 registers in SIC. Every register has an address associated with it known as a registration number. The size of each register is 3 bytes. On basis of register size, integer size is dependent.
    I. A(Accumulator-0): It is used for mathematical operations.
    II. X(Index Register-1): It is used for addressing.
    III. L(Linkage Register-2): It stores the return address of the instruction in case of subroutines.
    IV. PC(Program Counter-8): It holds the address of the next instruction to be executed.
    V. SW(Status Word-9): It contains a variety of information
  - Status Word Register:

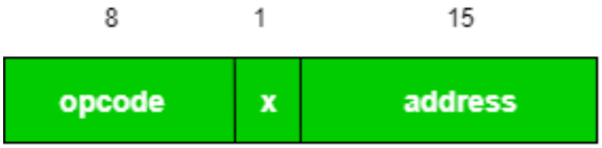
- **mode** bit refers to user mode(value=0) or supervising mode(value=1). It occupies 1 bit.
- **state** bit refers whether process is in running state(value=0) or idle state(value=1). 1 bit.
- **id** bit refers to process id(PID). It occupies 3 bits
- **CC** bit refers to condition code i.e. It tells whether the device is ready or not. It occupies 2 bits.
  Mask bit refers to interrupt mask. It occupies 4 bits.
- **X** refers to unused bit. It also occupies 4 bits.
- **ICode** refers to interrupt code i.e. Interrupt Service Routine. It occupies the remaining bits.

3. Data Format –

- Integers are represented by 24 bits.
- Negative numbers are represented in 2's complement.
- Characters are represented by 8 bit ASCII values.
- No floating-point representation is available.

4. Instruction Format –

All instructions in SIC have a 24-bit format.



- If x=0 it means direct addressing mode.
- If x=1 it means indexed addressing mode.

5. Instruction Set –

- Load And Store Instructions: To move or store data from accumulator to memory or vice-versa. For example LDA, STA, LDX, STX, etc.
- Comparison Instructions: Used to compare data in memory by contents in accumulator. For example COMP data.
- Arithmetic Instructions: Used to perform operations on accumulator and memory and store results in the accumulator. For example ADD, SUB, MUL, DIV, etc.
- Conditional Jump: compare the contents of accumulator and memory and performs task based on conditions. For example JLT, JEQ, JGT
- Subroutine Linkage: Instructions related to subroutines. For example JSUB, RSUB

6. Input and Output –

It is performed by transferring 1 byte at a time from or to the rightmost 8 bits of the accumulator. Each device has an 8-bit unique code.

There are 3 I/O instructions:

- Test Device (TD) tests whether the device is ready or not. Condition code in Status Word Register is used for this purpose. If cc is < then the device is ready otherwise the device is busy.
- Read data(RD) reads a byte from the device and stores it in register A.
- Write data(WD) writes a byte from register A to the device.

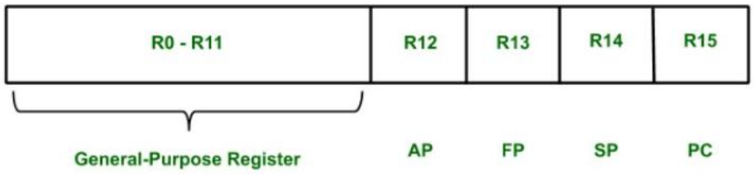
Here are some applications of SIC:

1. **Computer Architecture education:** The SIC is an excellent tool for teaching computer architecture and organization, as it provides a simplified model of a computer system. By studying the SIC's architecture, students can learn about the basic components of a computer system, such as the CPU, memory, and I/O devices.
2. **Assembly language programming education:** The SIC's instruction set is simple and easy to understand, making it a useful tool for teaching assembly language programming. Students can write and execute assembly language programs on the SIC, learning about the various instructions, addressing modes, and program flow control.
3. **Compiler development:** The SIC can be used as a platform for developing compilers for high-level programming languages. Compiler developers can use the SIC's instruction set and memory organization as a reference for generating assembly language code from high-level code.
4. **Operating system development:** The SIC's simple architecture can be used as a basis for teaching operating system development. Students can learn about the basic features of an operating system, such as process management, memory management, and I/O management, by implementing them on the SIC.
5. **Emulation and simulation:** The SIC can be used for emulation and simulation purposes, allowing software developers to test their programs on a simulated computer system before deploying them on real hardware.

VAX Architecture

VAX Architecture was designed to increase the compatibility by improving the hardware of the earlier designed machines. As VAX architecture is an example of the CISC (Complex Instruction Set Computers) therefore there are large and complicated instruction sets used in the system.

1. **Memory:** VAX architecture consists of 8- bit bytes memory. Two consecutive bytes form a word, four bytes form a longword, eight bytes form a quadword, sixteen bytes form an octaword. All VAX programs operates on **Virtual Address Space** (2^{32} bytes). The Virtual Address Space is divided into two spaces:
   - **System Space**
   - **Process Space**
2. **Registers:** VAX architecture have 16 general-purpose registers from R0 to R15. Some of these registers have special name and use.



AP - *Argument Pointer*

FP - *Frame Pointer*

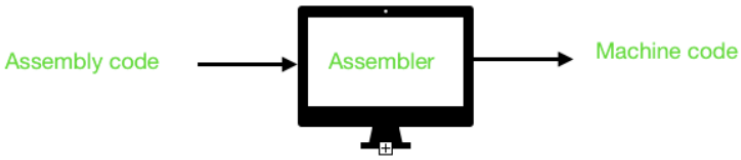
SP - *Stack Pointer*

PC - *Program Counter*

1. **Data Formats:**
   - Integers are stored as Binary numbers in byte, word, longword, quadword or octaword.
   - Characters are represented using 8-bit ASCII codes.
   - Floating points are represented using four different floating-point formats of length ranging from 4 to 16 bytes.
2. **Instruction Formats:** VAX machine architecture use a variable-length instruction format. Each instruction consists of an operand code (1 or 2 bytes) followed by up to six operand specifier, depending on the type of instruction.
3. **Addressing Modes:** VAX architecture use a large number of addressing modes. There are number of modes available such as register mode, register deferred mode, autoincrement and autodecrement mode. There are also base relative addressing modes, with displacement fields of different lengths. Program counter relative mode is also used to deal with PC register.
4. **Instruction Set:** In VAX systems instruction mnemonics are formed by combining following elements:
   - **Prefix:** A Prefix specifies the type of operation.
   - **Suffix:** A Suffix specifies the data type of the operands.
   - **Modifier:** A modifier specifies the number of operand involved.
5. **Input and Output:** I/O device controller are used to implement I/O on VAX architecture. Each controller has a set of control/status. The portion of the space into which the device controller register are mapped is called I/O space.

Introduction of Assembler

Assembler is a program for converting instructions written in low-level assembly code into relocatable machine code and generating along information for the loader.



It generates instructions by evaluating the mnemonics (symbols) in operation field and find the value of symbol and literals to produce machine code. Now, if assembler do all this work in one scan then it is called single pass assembler, otherwise if it does in multiple scans then called multiple pass assembler. Here assembler divide these tasks in two passes:

- **Pass-1:**
  1. Define symbols and literals and remember them in symbol table and literal table respectively.
  2. Keep track of location counter
  3. Process pseudo-operations
- **Pass-2:**
  1. Generate object code by converting symbolic op-code into respective numeric op-code
  2. Generate data for literals and look for values of symbols

Firstly, We will take a small assembly language program to understand the working in their respective passes. Assembly language statement format:

```
[Label] [Opcode] [operand]
```

Example:
```
M  ADD  R1, ='3'
```
where, M - Label; ADD - symbolic opcode;
R1 - symbolic register operand; (='3') - Literal

Assembly Program:

Label	Op-code	operand	LC value(Location counter)
JOHN	START	200	
	MOVER	R1, ='3'	200
	MOVEM	R1, X	201
L1	MOVER	R2, ='2'	202
	LTORG		203
X	DS	1	204
	END		205

Let's take a look on how this program is working:

1. **START:** This instruction starts the execution of program from location 200 and label with START provides name for the program.(JOHN is name for program)
2. **MOVER:** It moves the content of literal(='3') into register operand R1.
3. **MOVEM:** It moves the content of register into memory operand(X).
4. **MOVER:** It again moves the content of literal(='2') into register operand R2 and its label is specified as L1.
5. **LTORG:** It assigns address to literals(current LC value).
6. **DS(Data Space):** It assigns a data space of 1 to Symbol X.

7. **END:** It finishes the program execution.

Working of Pass-1: Define Symbol and literal table with their addresses.

Note: Literal address is specified by LTORG or END.

Step-1: START 200 (here no symbol or literal is found so both table would be empty)

Step-2: MOVER R1, ='3′ 200 (='3′ is a literal so literal table is made)

Literal	Address
='3′	– – –

Step-3: MOVEM R1, X 201

X is a symbol referred prior to its declaration so it is stored in symbol table with blank address field.

Symbol	Address
X	– – –

Step-4: L1 MOVER R2, ='2′ 202

L1 is a label and ='2′ is a literal so store them in respective tables

Symbol	Address
X	– – –
L1	202

Literal	Address
='3′	– – –
='2′	– – –

Step-5: LTORG 203

Assign address to first literal specified by LC value, i.e., 203

Literal	Address
='3′	203
='2′	– – –

Step-6: X DS 1 204

It is a data declaration statement i.e X is assigned data space of 1. But X is a symbol which was referred earlier in step 3 and defined in step 6.This condition is called Forward Reference Problem where variable is referred prior to its declaration and can be solved by back-patching. So now assembler will assign X the address specified by LC value of current step.

Symbol	Address
X	204
L1	202

Step-7: END 205

Program finishes execution and remaining literal will get address specified by LC value of END instruction. Here is the complete symbol and literal table made by pass 1 of assembler.

Symbol	Address
X	204
L1	202

Literal	Address
=’3′	203
=’2′	205

Now tables generated by pass 1 along with their LC value will go to pass-2 of assembler for further processing of pseudo-opcodes and machine op-codes.

Working of Pass-2:

Pass-2 of assembler generates machine code by converting symbolic machine-opcodes into their respective bit configuration(machine understandable form). It stores all machine-opcodes in MOT table (op-code table) with symbolic code, their length and their bit configuration. It will also process pseudo-ops and will store them in POT table(pseudo-op table).

Various Data bases required by pass-2:

1. MOT table(machine opcode table)
2. POT table(pseudo opcode table)
3. Base table(storing value of base register)
4. LC (location counter)

Different Architectures

The following section introduces the architectures of CISC and RISC machines. CISC machines are called traditional machines. In addition to these we have recent RISC machines. Different machines belonging to both of these architectures are compared with respect to their Memory, Registers, Data Formats, Instruction Formats, Addressing Modes, Instruction Set, Input and Output

CISC machines

Traditional (CISC) Machines, are nothing but, Complex Instruction Set Computers, has relatively large and complex instruction set, different instruction formats, different lengths, different addressing modes, and implementation of hardware for these computers is complex. VAX and Intel x86 processors are examples for this type of architecture.

VAX Architecture

Memory - The VAX memory consists of 8-bit bytes. All addresses used are byte addresses. Two consecutive bytes form a word, Four bytes form a longword, eight bytes form a quadword, sixteen bytes form a octaword. All VAX programs operate in a virtual address space of 232 bytes , One half is called system space, other half process space.

Registers – There are 16 general purpose registers (GPRs) , 32 bits each, named as R0 to R15, PC (R15), SP (R14), Frame Pointer FP (R13), Argument Pointer AP (R12) ,Others available for general use. There is a Process status longword (PSL) – for flags.

Data Formats - Integers are stored as binary numbers in byte, word, longword, quadword, octaword. 2’s complement notation is used for storing negative numbers. Characters are stored as 8-bit ASCII codes. Four different floating-point data formats are also available.

Instruction Formats - VAX architecture uses variable-length instruction formats – op code 1 or 2 bytes, maximum of 6 operand specifiers depending on type of instruction. Tabak – Advanced Microprocessors (2nd edition) McGraw-Hill, 1995, gives more information.

Addressing Modes - VAX provides a large number of addressing modes. They are Register mode, register deferred mode, autoincrement, autodecrement, base relative, program-counter relative, indexed, indirect, and immediate.

Instruction Set – Instructions are symmetric with respect to data type - Uses prefix – type of operation, suffix – type of operands, a modifier – number of operands. For example, ADDW2 - add, word length, 2 operands, MULL3 - multiply, longwords, 3 operands CVTCL - conversion from word to longword. VAX also provides instructionsto load and store multiple registers.

Input and Output - Uses I/O device controllers. Device control registers are mapped to separate I/O space. Software routines and memory management routines are used for input/output operations.

Pentium Pro Architecture

Introduced by Intel in 1995.

Memory - consists of 8-bit bytes, all addresses used are byte addresses. Two consecutive bytes form a word, four bytes form a double word (dword). Viewed as collection of segments, and, address = segment number + offset. There are code, data, stack , extra segments.

Registers – There are 32-bit, eight GPRs, namely EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP. EAX, EBX, ECX, EDX – are used for data manipulation, other four are used to hold addresses. EIP – 32-bit contains pointer to next instruction to be executed. FLAGS is an 32 - bit flag register. CS, SS, DS, ES, FS, GS are the six 16-bit segment registers.

Data Formats - Integers are stored as 8, 16, or 32 bit binary numbers, 2's complement for negative numbers, BCD is also used in the form of unpacked BCD, packed BCD. There are three floating point data formats, they are single, double, and extended- precision. Characters are stored as one per byte – ASCII codes.
Instruction Formats – Instructions uses prefixes to specify repetition count, segment register, following prefix (if present), an opcode (1 or 2 bytes), then number of bytes to specify operands, addressing modes. Instruction formats varies in length from 1 byte to 10 bytes or more. Opcode is always present in every instruction

Addressing Modes - A large number of addressing modes are available. They are immediate mode, register mode, direct mode, and relative mode. Use of base register, index register with displacement is also possible.

Instruction Set – This architecture has a large and complex instruction set, approximately 400 different machine instructions. Each instruction may have one, two or three operands. For example Register-to-register, register-to-memory, memory-to-memory, string manipulation, etc…are the some the instructions.

Input and Output - Input is from an I/O port into register EAX. Output is from EAX to an I/O port

RISC Machines

RISC means Reduced Instruction Set Computers. These machines are intended to simplify the design of processors. They have Greater reliability, faster execution and less expensive processors. And also they have standard and fixed instruction length. Number of machine instructions, instruction formats, and addressing modes relatively small. UltraSPARC Architecture and Cray T3E Architecture are examples of RISC machines.

Ultra SPARC Architecture

Introduced by Sun Microsystems. SPARC – Scalable Processor ARChitecture. SPARC, SuperSPARC, UltraSPARC are upward compatible machines and share the same basic structure.

Memory - Consists of 8-bit bytes, all addresses used are byte addresses. Two consecutive bytes form a halfword, four bytes form a word , eight bytes form a double word. Uses virtual address space of 2^{64} bytes, divided into pages.

Registers - More than 100 GPRs, with 64 bits length each called Register file. There are 64 double precision floating-point registers, in a special floating-point unit (FPU). In addition to these, it contains PC, condition code registers, and control registers.

Data Formats - Integers are stored as 8, 16, 32 or 64 bit binary numbers. Signed, unsigned for integers and 2's complement for negative numbers. Supports both big-endian and little-endian byte orderings. Floating-point data formats – single, double and quad-precision are available. Characters are stored as 8-bit ASCII value.

Instruction Formats - 32-bits long, three basic instruction formats, first two bits identify the format. Format 1 used for call instruction. Format 2 used for branch instructions. Format 3 used for load, store and for arithmetic operations.

Addressing Modes - This architecture supports immediate mode, register-direct mode,PC-relative, Register indirect with displacement, and Register indirect indexed.

Instruction Set – It has fewer than 100 machine instructions. The only instructions that access memory are loads and stores. All other instructions are register-to-register operations. Instruction execution is pipelined – this results in faster execution, and hence speed increases.

Input and Output - Communication through I/O devices is accomplished through memory. A range of memory locations is logically replaced by device registers. When a load or store instruction refers to this device register area of memory, the corresponding device is activated. There are no special I/O instructions.

Cray T3E Architecture

Announced by Cray Research Inc., at the end of 1995 and is a massively parallel processing (MPP) system, contains a large number of processing elements (PEs), arranged in a

three-dimensional network. Each PE consists of a DEC Alpha EV5 RISC processor, and local memory.

Memory - Each PE in T3E has its own local memory with a capacity of from 64 megabytes to 2 gigabytes, consists of 8-bit bytes, all addresses used are byte addresses. Two consecutive bytes form a word, four bytes form a longword, eight bytes form a quadword.

Registers – There are 32 general purpose registers(GPRs), with 64 bits length each called R0 through R31, contains value zero always. In addition to these, it has 32 floating-point registers, 64 bits long, and 64-bit PC, status , and control registers.

Data Formats - Integers are stored as long and quadword binary numbers. 2's complement notation for negative numbers. Supports only little-endian byte orderings. Two different floating-point data formats – VAX and IEEE standard. Characters stored as8-bit ASCII value.

Instruction Formats - 32-bits long, five basic instruction formats. First six bits always identify the opcode.

Addressing Modes - This architecture supports, immediate mode, register-direct mode, PC-relative, and Register indirect with displacement.

Instruction Set - Has approximately 130 machine instructions. There are no byte or word load and store instructions. Smith and Weiss – "PowerPC 601 and Alpha 21064: A Tale of TWO RISCs " – Gives more information.

Input and Output - Communication through I/O devices is accomplished through multiple ports and I/O channels. Channels are integrated into the network that interconnects the processing elements. All channels are accessible and controllable from all PEs.