Systems Programming

Chapter 1

Background

Outline

• Introduction to system software
• An overview of the material
• Relationship bw system software and machine architecture
• Simplified Instructional Computer (SIC)
• Architecture of several computers

Examples of System Software

• text editor: create/modify a program in a high level language
• compiler: translate it to machine code
• loader/linker: load resulting code into memory and prepare for execution
• debugger: detect errors in program

Examples of System Software

• text editor: create/modify a program in assembly language
• w/macro instructions to read and write data, or to perform other higher-level functions
• assembler/macro processor: translate it to machine code
• loader/linker: load resulting code into memory and prepare for execution
• debugger: detect errors in program

Examples of System Software

• Operating system
  - UNIX, DOS: textual user interface
  - MacOS, Windows: GUI (graphical user interface)

  - takes care of all machine-level details for us
  • connect to network, use different kinds of devices, perform input/output, etc.
We will . . .

• understand the processes that were going on “behind the scenes” as you used the computer
• gain a deeper understanding of how computers actually work

Machine-Independent Aspects

• Assembler
  - general design and logic
• Compilers
  - some code optimization techniques
• Linkers
  - linking independently assembled subprograms does not usually depend on the computer being used

System Software and Machine Architecture

System software
• machine dependent
• intended to support the operation and use of the computer itself

Application software
• machine independent
• intended to support a particular application

Machine-Dependent Aspects

• Assemblers:
  - translate mnemonic instructions into machine code
  - the instruction formats, addressing modes, etc. are of direct concern in assembler design
• Compilers
  - generate machine language code
  - number and type of registers and machine instructions available
• Operating Systems
  - manage the resources of a computer

Simplified Instructional Computer (SIC)

• a hypothetical computer that has been carefully designed to include the hardware features most often found in real machines
  - unusual or irrelevant complexities of real machines have been avoided

Organization of Chapters

• Fundamental features of system software being discussed
• Machine-dependent features
• Machine-independent features
• Major design options for structuring a particular piece of software
• Examples of implementations on actual machines
Simplified Instructional Computer (SIC)

- Two versions
  - standard model
  - XE model
- They are upward-compatible: An object program for the standard SIC machine will also execute properly on a SIC/XE system.

QUESTION

- How can you characterize a machine architecture?

SIC Machine Architecture

- Memory
- Registers
- Data formats
- Instruction formats
- Addressing modes
- Instruction set
- Input and Output

- Memory:
  - \(2^{15}\) bytes, 3-byte words, byte addresses
- Registers:
  - 5 3-byte long registers

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Number</th>
<th>Special Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>Accumulator; used for arithmetic operations</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>Index register; used for addressing</td>
</tr>
<tr>
<td>L</td>
<td>2</td>
<td>Linkage register; the Jump to Subroutine (JSUB) instruction stores the return address</td>
</tr>
<tr>
<td>PC</td>
<td>8</td>
<td>Program Counter; the address of next instruction to be fetched</td>
</tr>
<tr>
<td>SW</td>
<td>9</td>
<td>Status Word; contains a variety of information, including a Condition Code (CC)</td>
</tr>
</tbody>
</table>

- Data formats:
  - 24-bit Integers (2's complement form for negatives)
  - 8-bit ASCII chars

- Instruction formats:
  - 24-bit instructions

<table>
<thead>
<tr>
<th>Opcode</th>
<th>x</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>1</td>
<td>15</td>
</tr>
</tbody>
</table>

- Addressing modes:

<table>
<thead>
<tr>
<th>Mode</th>
<th>Indication</th>
<th>Target Address Calculation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct</td>
<td>x = 0</td>
<td>TA = address</td>
</tr>
<tr>
<td>Indexed</td>
<td>x = 1</td>
<td>TA = address + (X)</td>
</tr>
</tbody>
</table>

- Instruction Set:
  - Load and store registers: LDA, LDX, STA, STX, etc.
  - Integer arithmetic operations: ADD, SUB, MUL, DIV
    - all involve register A and a word in memory, with the result left in the register
  - COMP: compare the value in A with a word in memory, set the condition code (CC) to indicate the result (<, =, >)
  - Conditional jump instructions: JLT, JEQ, JGT
  - Subroutine linkage:
    - JSUB jumps to subroutine, places return address in register L
    - RSUB returns by jumping to the address contained in register L
• **Input and Output:**
  - performed by transferring 1 byte at a time to or from the rightmost 8 bits of register A
  - Each device is assigned a unique 8-bit code.
  - Test Device (TD) instruction tests whether the addressed device is ready to send or receive a byte of data and sets CC (\( \neq \) ready)
  - Read Data (RD) reads a byte
  - Write Data (WD) writes a byte

---

**SIC/XE Machine Architecture**

- More Memory
- Extra Registers
- Data formats - floating point numbers
- Additional Instruction formats
- More Addressing modes
- Extended Instruction set
- Input and Output - channels to overlap I/O and processing

---

**SIC Sample Data Movement Operations**

- there are no memory-to-memory move instructions
  - `LDA FIVE` load constant 5 into register A
  - `STC C1` store in character variable C1
  - `CHRC CHARZ` one-byte constant

  Four ways of defining storage

---

**SIC Sample Arithmetic Operations**

- All arithmetic operations are performed using register A, with the result being left in register A.
  - `LDA ALPHA` load alpha into register A
  - `ADD INCR` increment the value in register A
  - `SUB ONE` subtract one
  - `STA BETA` store in beta

---

**SIC Looping and Indexing Operations**

- `LDA ZERO` initialize index register to 0
- `MOVECH LDCH STR1,X` load character from STR1 into reg A
- `STCH STR2,X` store character into STR2
- `TIX ELEVEN` add 1 to index, compare result to 11
- `JLT MOVECH` loop if index is less than 11
- `ADD INCR` add the value of increment

---

**SIC Looping and Indexing Operations 2**

- Add together corresponding elements of ALPHA and BETA, store results in GAMMA.
  - `LDA ZERO` initialize index value to 0
  - `ADDLP LDX INDEX` load index value into reg x
  - `LDA ALPHA,X` load word from alpha into reg a
  - `STA INDEX` store the result in a word in gamma

---

**SIC Sample Arithmetic Operations**

- All arithmetic operations are performed using register A, with the result being left in register A.
  - `LDA ALPHA` load alpha into register A
  - `ADD INCR` increment the value in register A
  - `SUB ONE` subtract one
  - `STA BETA` store in beta

---

**SIC Loops and Indexing Operations**

- `LDA ZERO` initialize index register to 0
- `MOVECH LDCH STR1,X` load character from STR1 into reg A
- `STCH STR2,X` store character into STR2
- `TIX ELEVEN` add 1 to index, compare result to 11
- `JLT MOVECH` loop if index is less than 11
- `ADD INCR` add the value of increment

---

**SIC Sample Arithmetic Operations**

- All arithmetic operations are performed using register A, with the result being left in register A.
  - `LDA ALPHA` load alpha into register A
  - `ADD INCR` increment the value in register A
  - `SUB ONE` subtract one
  - `STA BETA` store in beta

---

**SIC Loops and Indexing Operations 2**

- Add together corresponding elements of ALPHA and BETA, store results in GAMMA.
  - `LDA ZERO` initialize index value to 0
  - `ADDLP LDX INDEX` load index value into reg x
  - `LDA ALPHA,X` load word from alpha into reg a
  - `STA INDEX` store the result in a word in gamma

---

Four ways of defining storage
SIC Input/Output Operations

• Read 1 byte of data from device F1 and copy it to device 05.

INLOOP TO INDEV TEST INPUT DEVICE
JEQ INLOOP LOOP UNTIL DEVICE IS READY
RD INDEV READ ONE BYTE INTO REGISTER A
STCH DATA STORE BYTE THAT WAS READ

OUTLP TO OUTDEV TEST OUTPUT DEVICE
JEQ OUTLP LOOP UNTIL DEVICE IS READY
LDCH DATA LOAD DATA BYTE INTO REGISTER A
WD OUTDEV WRITE ONE BYTE TO OUTPUT DEVICE

INDEV BYTE X’F1’ INPUT DEVICE NUMBER
OUTDEV BYTE X’05’ OUTPUT DEVICE NUMBER
DATA RESB 1 ONE-BYTE VARIABLE

RISC (Reduced Instruction Set Computers)

• Developed in early 1980s to simplify the design of processors
  - faster, less expensive processors
  - greater reliability
  - faster instruction execution times
• Fixed instruction length (usually 1 word)
  - Single cycle execution for most instructions
• All instructions, except Load and Store, are register-to-register
  - Memory access by Load and Store instrs only
  - Have large number of general purpose registers
• Number of machine instructions, instruction formats, and addressing modes are small.
• Examples: UltraSPARC, PowerPC.

SIC Subroutine Call and Record Input Operations

• Read a 100-byte record from an input device into memory.

JSUB READ CALL READ ROUTINE

READ LDX ZERO SUBROUTINE TO READ 100-BYTE RECORD
RLOOP TO INDEV TEST INPUT DEVICE
JEQ RLOOP LOOP UNTIL DEVICE IS READY
RD INDEV READ ONE BYTE INTO REGISTER A
STCH RECORD,X STORE DATA BYTE INTO RECORD
TIX K100 ADD 1 TO INDEX AND COMPARE TO 100
JLT RLOOP LOOP IF INDEX IS LESS THAN 100
RSUB EXIT FROM SUBROUTINE

INDEV BYTE X’F1’ INPUT DEVICE NUMBER
RECORD RESB 100 100-BYTE BUFFER FOR INPUT RECORD
ZERO WORD 0
K100 WORD 100

Main Machine Architectures

• CISC (Complex Instruction Set Computers)
  - relatively large and complicated instruction set
  - several different instruction formats and lengths
  - many different addressing modes
  - requires complicated hardware mechanisms
  - Examples: VAX, Pentium Pro
• RISC (Reduced Instruction Set Computers)