CSE 410
Operating Systems

Today’s Lecture

- Course organization
- Computing environment
- Overview of course topics
Course Organization

- Course website
  
  http://www.cse.msu.edu/~cse410/

- Syllabus and calendar

- Enrollment

Computing Environment

- Server: cse410.cse.msu.edu

- Operating system: Linux (Debian 8)

- Accounts
Operating Systems Overview

Operating system: software that serves as the interface between application programs and computer hardware

Manages resources (hardware):
- Processor
- Primary storage
- Secondary storage

Example: Linux Overview
Example: Linux Overview

Review: Components of a Computer

- Processor (CPU)
- Primary storage (RAM)
- I/O module
- System interconnection (bus)
Figure 1.1  Computer Components: Top-Level View

PC  =  Program counter
IR  =  Instruction register
MAR  =  Memory address register
MBR  =  Memory buffer register
I/O AR  =  Input/output address register
I/O BR  =  Input/output buffer register
Pentium

Instruction Cycle

CPU fetches one instruction, executes it

Fetch Cycle

Execute Cycle

Basic Instruction Cycle
The Fetch-Execute Cycle

Fetch Phase:
- \( \text{RAM} [ \text{PC} ] \rightarrow \text{IR} \)

Execute Phase:
- decode IR
- take appropriate action
- update PC

Example: ARM microprocessor
- Assume each instruction is 4 bytes long
- Assume PC: 00010700
- Fetch phase:
  - access RAM[ 00010700 ]
  - copy 4 bytes (E0861007) to IR
- IR now contains: E0861007
Example (continued)

- IR now contains: E0861007
- Execute phase:
  - decode IR
    - ADD instruction on ARM
  - take appropriate action
  - update PC
    - PC + 4 ==> PC

Example (continued)

<table>
<thead>
<tr>
<th>RAM Address</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td></td>
</tr>
<tr>
<td>00000001</td>
<td></td>
</tr>
<tr>
<td>00000002</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>00010700</td>
<td>E0</td>
</tr>
<tr>
<td>...</td>
<td>86</td>
</tr>
<tr>
<td>...</td>
<td>10</td>
</tr>
<tr>
<td>...</td>
<td>07</td>
</tr>
<tr>
<td>FFFFFFFF</td>
<td></td>
</tr>
<tr>
<td>FFFFFFFF</td>
<td></td>
</tr>
<tr>
<td>FFFFFFFF</td>
<td></td>
</tr>
</tbody>
</table>

CPU

- PC: 00010700
- IR: E0861007
Example (continued)

Example: C/C++ Program

From www.cse.msu.edu/~cse410/Examples/example01

```c
int sum = 0;

int main()
{
    for( int i = 1; i <= 6; i++)
    {
        sum = sum + i;
    }
    return 0;
}
```
Equivalent ARM Assembly Language

.globl main
.text
main:  push  {lr}
       mov   r0, #1
loop:  cmp   r0, #6
       bgt   end
       ldr   r2, =sum
       ldr   r1, [r2]
       add   r1, r1, r0
       str   r1, [r2]
       add   r0, r0, #1
       b     loop
end:   pop   {lr}
       mov   pc, lr

Equivalent ARM Machine Language

.globl main
.text
0000 E52DE004  main:   push  {lr}
0004 E3A00001  mov   r0, #1
0008 E3500006  loop:  cmp   r0, #6
000c CA000005  bgt   end
0010 E59F2018  ldr   r2, =sum
0014 E5921000  ldr   r1, [r2]
0018 E0811000  add   r1, r1, r0
001c E5821000  str   r1, [r2]
0020 E2800001  add   r0, r0, #1
0024 EAFFFFF7  b     loop
0028 E49DE004  end:   pop  {lr}
002c E1A0F00E  mov   pc, lr
Execution Trace

<table>
<thead>
<tr>
<th>Time</th>
<th>PC</th>
<th>IR</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00010800</td>
<td>E52DE004</td>
<td>main:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>push {lr}</td>
</tr>
<tr>
<td>1</td>
<td>00010804</td>
<td>E3A00001</td>
<td>mov r0, #1</td>
</tr>
<tr>
<td>2</td>
<td>00010808</td>
<td>E35000006</td>
<td>loop:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>cmp r0, #6</td>
</tr>
<tr>
<td>3</td>
<td>0001080c</td>
<td>CA000005</td>
<td>bgt end</td>
</tr>
<tr>
<td>4</td>
<td>00010810</td>
<td>E59F2018</td>
<td>ldr r2, =sum</td>
</tr>
<tr>
<td>5</td>
<td>00010814</td>
<td>E5921000</td>
<td>ldr r1, [r2]</td>
</tr>
<tr>
<td>6</td>
<td>00010818</td>
<td>E0811000</td>
<td>add r1, r1, r0</td>
</tr>
<tr>
<td>7</td>
<td>0001081c</td>
<td>E5821000</td>
<td>str r1, [r2]</td>
</tr>
<tr>
<td>8</td>
<td>00010820</td>
<td>E2800001</td>
<td>add r0, r0, #1</td>
</tr>
<tr>
<td>9</td>
<td>00010824</td>
<td>EAFFFF7</td>
<td>b loop</td>
</tr>
<tr>
<td>10</td>
<td>00010808</td>
<td>E35000006</td>
<td>loop:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>cmp r0, #6</td>
</tr>
<tr>
<td>11</td>
<td>0001080c</td>
<td>CA000005</td>
<td>bgt end</td>
</tr>
</tbody>
</table>

CPU Registers

- High-speed storage closely tied to CPU
- General Purpose (User-Visible) Registers
  - available to both OS and user programs
  - hold operands, results, and addresses
- Control and Status Registers
  - not directly available to user programs
  - used by CPU to control its own operation and to control program execution
Control & Status Registers

- Program Counter (PC)
  - address of the next instruction to be fetched

- Instruction Register (IR)
  - instruction most recently fetched

- Program Status Word (PSW)
  - condition codes bits (zero flag, sign flag, etc.)
  - supervisor/user mode bit
  - interrupt enable/disable bit

Types of Instructions

- Data Movement: transfer data from RAM to CPU (load) or from CPU to RAM (store)

- Data Manipulation: perform arithmetic or logic operation on data

- Control Transfer: alter the execution sequence of the program

- Other: special purpose
Privileged Instructions

- Can only be executed by the OS (user cannot execute privileged instructions)
  - ex: initialize PC
  - ex: place value in PSW
- Dual-mode operation: status bit for user mode or kernel (supervisor) mode
- Some processors support more than two modes (additional status bits)

Interrupts

- Fundamental to operation of modern computers and integral part of OS design
- Mechanism to allow other modules to interrupt CPU
  - ex: I/O device signals that operation is complete
  - ex: Timer signals that quantum is over
- Interrupts can occur at any time
Modified Instruction Cycle

CPU checks for interrupts after each instruction, invokes interrupt handler

Examples of Interrupts

- Hardware failure (ex: memory parity error)
- Timer (ex: time slice completed)
- I/O (ex: operation completed)
- Program Exception (ex: divide by zero)
- Program Trap (control transfer instruction to request interrupt)
Interrupt Handler

- Code that determines nature of the interrupt and performs necessary actions

- State of the interrupted process must be saved so the process can be resumed later as if nothing had happened

- Interrupt handler addresses (interrupt vector table) configured at boot time

Interrupt Processing
Interrupt Processing

- The *state* of a process is all of the bit patterns related to that process:
  - machine language instructions in RAM
  - data objects in RAM
  - contents of CPU registers

- An interrupt handler must preserve the state of the process which was interrupted (leave RAM alone, save contents of registers)
Simple Interrupt Processing

- Interrupt handling involves hardware and software
- The interrupted program is unaware of the interrupt
- Interrupt handling should be fast.

Interrupt Categories

- Hardware interrupt – signal from device
- Software interrupt – signal from program which is currently executing
  - Exception – error caused by current instruction
  - Trap – specific instruction to cause interrupt in a controlled way