Operating Systems

Prof. Christophe Bobda
CSCE-Department
University of Arkansas
Chapter 0
Introduction
Agenda

• **Computer System Structure**
  • Computer systems,
  • Application Specific Processors,
  • Von Neumann Computers
  • System Integration

• **What is an Operating System**
  • Structure, declaration, compilation, execution, simulation

• **Syllabus**
Computer Systems
Ubiquitous and Pervasive
Computing Paradigms
Application Specific Processors (ASP, ASIP)

Array $A, B: [1:n]$
Real $c = A \times B$;

$C = 0$;
for $i=1$ to $n$ do

$C = C + A[i] \times B[i]$;

End for;

• Maximal Performance
• Minimal Flexibility
1. Computer Paradigms

Von Neumann Computers

- Maximal Flexibility
- Minimal Performance
# Instruction Set

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Function</th>
<th>Encoding</th>
<th>op</th>
<th>funct</th>
</tr>
</thead>
<tbody>
<tr>
<td>add $1, $2, $3</td>
<td>addition: $1 → $2 + $3</td>
<td>R</td>
<td>000000</td>
<td>100000</td>
</tr>
<tr>
<td>sub $1, $2, $3</td>
<td>subtraction: $1 → $2 − $3</td>
<td>R</td>
<td>000000</td>
<td>100010</td>
</tr>
<tr>
<td>and $1, $2, $3</td>
<td>bitwise and: $1 → $2 and $3</td>
<td>R</td>
<td>000000</td>
<td>100100</td>
</tr>
<tr>
<td>or $1, $2, $3</td>
<td>bitwise or: $1 → $2 or $3</td>
<td>R</td>
<td>000000</td>
<td>100101</td>
</tr>
<tr>
<td>slt $1, $2, $3</td>
<td>set less than: $1 → 1 if $2 &lt; $3</td>
<td>R</td>
<td>000000</td>
<td>101010</td>
</tr>
<tr>
<td>addi $1, $2,</td>
<td>add immediate: $1 → $2 + imm</td>
<td>I</td>
<td>001000</td>
<td>n/a</td>
</tr>
<tr>
<td>beq $1, $2, imm</td>
<td>branch if equal: PC → PC + imm$^a$</td>
<td>I</td>
<td>000100</td>
<td>n/a</td>
</tr>
<tr>
<td>j destination</td>
<td>jump: PC_destination$^a$</td>
<td>J</td>
<td>000010</td>
<td>n/a</td>
</tr>
<tr>
<td>lb $1, imm($2)</td>
<td>load byte: $1 → mem[$2 + imm]</td>
<td>I</td>
<td>100000</td>
<td>n/a</td>
</tr>
<tr>
<td>sb $1, imm($2)</td>
<td>store byte: mem[$2 + imm] → $1</td>
<td>I</td>
<td>110000</td>
<td>n/a</td>
</tr>
</tbody>
</table>
## Instruction Encoding

- **32-bit instruction encoding**
  - Requires four cycles to fetch on 8-bit datapath

<table>
<thead>
<tr>
<th>format</th>
<th>example</th>
<th>encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>add $rd, $ra, $rb</td>
<td>6 5 5 5 0 6 funct</td>
</tr>
<tr>
<td>I</td>
<td>beq $ra, $rb, imm</td>
<td>6 5 5 16</td>
</tr>
<tr>
<td>J</td>
<td>j dest</td>
<td>6 26</td>
</tr>
</tbody>
</table>

**Example Encoding**:

- **R** format example:
  - `add $rd, $ra, $rb`:
    - Encoding:
      - **R** format
      - `0 ra rb rd 0 funct`:
        - **0**: Value
        - **ra**: Register A
        - **rb**: Register B
        - **rd**: Register D
        - **0**: Value
        - **funct**: Function

- **I** format example:
  - `beq $ra, $rb, imm`:
    - Encoding:
      - **I** format
      - `op ra rb imm`:
        - **op**: Operation
        - **ra**: Register A
        - **rb**: Register B
        - **imm**: Immediate

- **J** format example:
  - `j dest`:
    - Encoding:
      - **J** format
      - `op dest`:
        - **op**: Operation
        - **dest**: Destination
MIPS Microarchitecture

- Multicycle architecture from Patterson & Hennessy
Multicycle Controller

Diagram showing the stages of execution:

1. Instruction fetch
2. Instruction decode/register fetch
3. Memory address computation
4. Execution
5. Branch completion
6. Jump completion
7. Write-back step
8. Memory access
9. Memory access
10. Memory access
11. Memory access
12. Memory access

Stages:
- MemRead
- ALUSrcA = 0
- IRWrite3
- ALUSrcB = 01
- ALUOp = 00
- PCWrite
- PCSource = 00

- MemRead
- ALUSrcA = 0
- IRWrite3
- ALUSrcB = 01
- ALUOp = 00
- PCWrite
- PCSource = 00

- MemRead
- ALUSrcA = 0
- IRWrite3
- ALUSrcB = 01
- ALUOp = 00
- PCWrite
- PCSource = 00

- MemRead
- ALUSrcA = 0
- IRWrite3
- ALUSrcB = 01
- ALUOp = 00
- PCWrite
- PCSource = 00

- MemRead
- ALUSrcA = 0
- IRWrite3
- ALUSrcB = 01
- ALUOp = 00
- PCWrite
- PCSource = 00

- MemRead
- ALUSrcA = 0
- IRWrite3
- ALUSrcB = 01
- ALUOp = 00
- PCWrite
- PCSource = 00

- RegDst = 1
- RegWrite
- MemtoReg = 0

- RegDst = 1
- RegWrite
- MemtoReg = 0
Computer Systems Integration

Processor
  Cache

Interrupts

Main Memory

I/O Controller
  Disk

I/O Controller
  Graphics

I/O Controller

I/O Controller

Network

Memory-I/O Bus
Computer Systems
Layered Representation (Design)

Software
- High-Level Programming Languages
- Assembler
- Instructions Set Architecture (ISA)
- Micro Architecture (Register-Transfer-Level)

Hardware
- Logic Gates
- Transistors
- Geometry
Computer Systems
Layered Representation (Run-Time)

Software
- Applications
- Operating System
- Instructions Set Architecture (ISA)

Hardware
- Micro Architecture (Register-Transfer-Level)
- Logic Gates
- Transistors
- Geometry
What is an Operating System?

• Two Main Views to Characterize OSes
  • Service Provider
  • Resource Manager
### Parallel Port Example

#### LPT Connection Pin | I/O Direction | Register Address
--- | --- | ---
1 | Output | 0th bit of register #2
2-9 | Output | 0th bit of register #2
10,11,12,13,15 | Input | 6,7,5,4,3th bit of register #1
14,16,17 | Output | 1,2,3th bit of register #2

- Using assembly language program the printer
  - Interaction with the printer over a set of registers

- Printer routine will roughly consist of
  - Initialize the printer
  - Copy the data to print in memory
  - Successively copy the data from memory to corresponding printer

Control the paper,
Parallel Port Example

- Printer routine will roughly consist of
  - Initialize the printer
    - Check if printer is on
    - Check if printer has paper
    - Check if printer has ink/toner
  - Gain control of the printer head and start the printer
  - Copy the data to print in memory
  - Successively copy the data from memory to corresponding printer and wait for the printer to acknowledge the printing
  - Constantly check the printer status
    - Paper jam
    - Ink issues
This program consists of a sub-routine that reads the state of the input pin, determining the on/off state of our switch and asserts the output pin, turning the LED on/off accordingly.

```assembly
CheckPort proc
push ax ; save the content
push dx ; save the content
mov dx, 3BCh + 1 ; base + 1 for register #1
in al, dx ; read register #1
and al, 10h ; mask out all but bit # 4
cmp al, 0 ; is it 0?
jne SwitchOn ; if not, we need to turn the LED on

SwitchOff:
mov dx, 3BCh + 0 ; base + 0 for register #0
in al, dx ; read the current state of the port
and al, f7h ; clear first bit (masking)
out dx, al ; write it out to the port
jmp Done ; we are done

SwitchOn:
mov dx, 3BCh + 0 ; base + 0 for register #0
in al, dx ; read the current state of the port
or al, 01h ; set first bit (masking)
out dx, al ; write it out to the port

Done: pop dx ; restore the content
pop ax ; restore the content
CheckPort endp
```

extern “C” CheckPort(void); // defined in assembly

```c
void main(void) {
    while( 1 ) {
        CheckPort();
    }
}
```

<table>
<thead>
<tr>
<th>LPT Connection Pin</th>
<th>I/O Direction</th>
<th>Register Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Output</td>
<td>0(^{th}) bit of register #1</td>
</tr>
<tr>
<td>2-9</td>
<td>Output</td>
<td>0(^{th}) bit of register #2</td>
</tr>
<tr>
<td>10,11,12,13,15</td>
<td>Input</td>
<td>6,7,5,4,3(^{th}) bit of register #1</td>
</tr>
<tr>
<td>14,16,17</td>
<td>Output</td>
<td>1,2,3(^{th}) bit of register #2</td>
</tr>
</tbody>
</table>
OS as Service Provider

Application 1
Resource access 1

Application 2
Resource access 2

Application n
Resource access n

Resource
OS as Service Provider

Application 1

Application 2

Application n

Resource access

Resource
OS as Service Provider

• It is an extended machine
  • Hides the messy details which must be performed
  • Presents user with a virtual machine, easier to use

• Advantage
  • Performance
  • Modularity
  • Memory usage
  • Time to market
  • Reliability
  • Maintainability
  • Portability
  • Scalability
OS as Resource Manager

Application 1  Application 2  Application n

Resource Management

Resources (Hardware)
**OS as Resource Manager**

- Each program gets time with the resource
- Each program gets space on the resource

**Task**

- Provide the environment under which program can run
- Control and manage the access to resources by concurrent programs
  - Processor management
  - Memory management
- Provide protection and security
Syllabus

• **Course Goal**
  • To learn the general concepts and architectures of OSes
  • To learn the design challenges of OSes
  • Understand the key components of OSes as well as their interactions
  • Understand the interaction between the OS and the HW
  • Attend the class frequently, solve the home assignment and make the instructor happy
Syllabus

• Lectures & Exercise:
  • Mon, Wed, Fri 10:30 AM – 11:20 AM, JBHT0147

• Office Hour
  • Monday 4 – 6 PM
  • JBHT0528
  • Tel: 575-4797
  • Email: cbobda@uark.edu
Syllabus

• Literature
  • Modern Operating Systems (3rd Edition)
    Andrew S. Tanenbaum
  
  • Operating System Concepts
    Abraham Silberschatz, Peter B. Galvin, Greg Gagne
Syllabus

- **Time Table (Tentative)**
  - 01.19.2011 - Introduction
  - 01.21.2011 - OS-Concepts & Structure
  - 01.21.2011 - OS-Concept & Structure
  - 01.24.2011 - OS-Concept & Structure
  - 01.26.2011 - Process & Threads
  - 01.28.2011 - Process & Threads
  - 01.31.2011 - Process & Threads
  - 02.02.2011 - CPU Scheduling
  - 02.04.2011 - CPU Scheduling
  - 02.07.2011 - Process Synchronization
Syllabus

• Time Table (Tentative)
  • 02.09.2011 - Open
  • 02.11.2011 - Process Synchronization
  • 02.14.2011 - Process Synchronization
  • 02.16.2011 - Deadlocks
  • 02.18.2011 - Deadlocks
  • 02.21.2011 - Deadlocks
  • 02.23.2011 - Memory Management
  • 02.25.2011 - Memory Management
  • 02.28.2011 - Memory Management
  • 03.02.2011 - Virtual Memory
Syllabus

• Time Table (Tentative)
  • 03.04.2011 - Virtual Memory
  • 03.07.2011 - Virtual Memory
  • 03.09.2011 - Virtual Memory
  • 03.11.2011 - File Systems
  • 03.14.2011 - File Systems
  • 03.16.2011 - File Systems
  • 03.18.2011 - File Systems
  • 03.21.2011 - File Systems
  • 03.23.2011 - File Systems
  • 03.25.2011 - I/O Systems
Syllabus

• Time Table (Tentative)
  • 03.28.2011 – Protection & Security
  • 03.30.2011 – Protection & Security
  • 04.01.2011 – Distributed Systems
  • 04.04.2011 – Distributed Systems
  • 04.06.2011 – Distributed Systems
  • 04.08.2011 – Distributed File Systems
  • 04.11.2011 – Multimedia Systems
  • 04.13.2011 – Real-Time Systems
  • 04.15.2011 – Real-Time Systems
  • 04.18.2011 – Case Studies
Syllabus

• **Time Table (Tentative)**
  • 04.20.2011 - Case Studies
  • 04.22.2011 - Case Studies
  • 04.25.2011 - Case Studies
  • 04.27.2011 - Case Studies
  • 05.02.2011 - Open
  • 05.04.2011 - Open
  • 05.13.2011 - Open
Syllabus

• Grading
  • 20% Home Assignment
  • 20% Projects
  • 60% Final Examination
  • 5% Class Participation (to be used for upgrade)
  • Final Grade = Sum of “subgrades”

<table>
<thead>
<tr>
<th>Percentage</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>95% - 100%</td>
<td>A+</td>
</tr>
<tr>
<td>90% - 94.9%</td>
<td>A</td>
</tr>
<tr>
<td>85% - 89.9%</td>
<td>A-</td>
</tr>
<tr>
<td>80% - 84.9%</td>
<td>B+</td>
</tr>
<tr>
<td>75% - 79.9%</td>
<td>B</td>
</tr>
<tr>
<td>70% - 74.9%</td>
<td>B-</td>
</tr>
<tr>
<td>65% - 69.9%</td>
<td>C+</td>
</tr>
<tr>
<td>60% - 64.9%</td>
<td>C</td>
</tr>
<tr>
<td>50% - 59.9%</td>
<td>C-</td>
</tr>
<tr>
<td>Below 50%</td>
<td>D or F</td>
</tr>
</tbody>
</table>
Syllabus

• Course Material
  • https://hthreads.csce.uark.edu/wiki/3613_Spring_11
    - Course slides
    - Home assignment
    - Current information and documents

• Simulators for projects
  - vip.cs.utsa.edu/simulators