CENG 311
Computer Architecture

Lecture 1

Introduction to Computer Architecture

Asst. Prof. Tolga Ayav, Ph.D.

Department of Computer Engineering
İzmir Institute of Technology
Few people design computers! Very few design instruction sets!
Many people design computer components.
Very many people are concerned with computer function, in detail.
The Big Picture

- What is inside a computer?
- How does it execute my program?
The Big Picture

• The Five Classic Components of a Computer
System Organization

- Processor
- Cache
- Memory Bus
- I/O Bridge
- Core Chip Set
- Main Memory
- I/O Bus
  - Disk Controller
  - Graphics Controller
  - Network Interface
  - Disk
  - Disk
  - Graphics
  - Network
What is Computer Architecture?

• Coordination of levels of abstraction

• Under a set of rapidly changing Forces

Software

Interface Between HW and SW

Hardware

Instruction Set Architecture, Memory, I/O
Levels of Representation

High Level Language Program

Compiler

Assembly Language Program

Assembler

Machine Language Program

Machine Interpretation

Control Signal Specification

temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;

lw $15, 0($2)
lw $16, 4($2)
sw $16, 0($2)
sw $15, 4($2)

0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111

İzmir Institute of Technology

Computer Architecture
Compiler

Assembler

High-level language program (in C)

```c
swap(int v[], int k)
{ int temp;
    temp = v[k];
    v[k] = v[k+1];
    v[k+1] = temp;
}
```

Assembly language program (for MIPS)

```
swap:
muli $2, $5, 4
add $2, $4, $2
lw $15, 0($2)
lw $16, 4($2)
sw $16, 0($2)
sw $15, 4($2)
jr $31
```
Translation hierarchy for C

C program
---
Compiler
---
Assembly language program
---
Assembler
---
Object: Machine language module
---
Linker
---
Object: Library routine (machine language)
---
Linker
---
Executable: Machine language program
---
Loader
---
Memory
Basic Elements

Functional Levels:
- Application Layer
- System Software
- Hardware Layer
MIPS Assembly

- move $t0, $t1
- add $t0, $zero, $t1
- sll $t1, $a1, 2 \ (\text{reg } t1=k*4)\n- lw $t0=4($t1) \ (\text{reg } t0=v[k+1])\n- ...
EPROM as a Programmable Logic Device

• ROMs are required for applications in which large amount of information needs to be stored in a nonvolatile manner. (Storage for microprocessor programs, fixed table of data, etc.) Another common application of the ROM is for the systematic realization of complex combinational circuits.
A field-programmable gate array is a semiconductor device containing programmable logic components called "logic blocks", and programmable interconnects. Logic blocks can be programmed to perform the function of basic logic gates such as AND, and XOR, or more complex combinational functions such as decoders or simple mathematical functions. In most FPGAs, the logic blocks also include memory elements, which may be simple flip-flops or more complete blocks of memory.

A classic FPGA logic block consists of a 4-input lookup table (LUT), and a flip-flop:
A soft microprocessor (also called softcore microprocessor or a soft processor) is a microprocessor core that can be wholly implemented using logic synthesis. It can be implemented via different semiconductor devices containing programmable logic (e.g., FPGA, CPLD).

Notable soft microprocessors include:

- **MicroBlaze**
- **Nios II**

<table>
<thead>
<tr>
<th>Processor</th>
<th>Developer</th>
<th>Open Source</th>
<th>Bus Support</th>
<th>Notes</th>
<th>Project Home</th>
</tr>
</thead>
<tbody>
<tr>
<td>MicroBlaze</td>
<td>Xilinx</td>
<td>no</td>
<td>OPB, FSL, LMB</td>
<td></td>
<td>Xilinx MicroBlaze</td>
</tr>
<tr>
<td>PicoBlaze</td>
<td>Xilinx</td>
<td>no</td>
<td></td>
<td></td>
<td>Xilinx PicoBlaze</td>
</tr>
<tr>
<td>Nios, Nios II</td>
<td>Altera</td>
<td>no</td>
<td></td>
<td></td>
<td>Altera Nios II</td>
</tr>
<tr>
<td>Cortex-M1</td>
<td>Arm</td>
<td>no</td>
<td></td>
<td></td>
<td>[1]</td>
</tr>
<tr>
<td>Micc32</td>
<td>Lattice</td>
<td>yes</td>
<td></td>
<td></td>
<td>LatticeMicc32</td>
</tr>
<tr>
<td>AEMB</td>
<td>Shawn Tan</td>
<td>yes</td>
<td>Wishbone</td>
<td>MicroBlaze EDK 3.2 compatible Verilog core</td>
<td>AEMB</td>
</tr>
<tr>
<td>OpenFire</td>
<td>Virginia Tech CCM Lab</td>
<td>yes</td>
<td>OPB, FSL</td>
<td>Binary compatible with the MicroBlaze</td>
<td>VT OpenFire</td>
</tr>
<tr>
<td>PacoBlaze</td>
<td>Pablo Bleyer</td>
<td>yes</td>
<td></td>
<td>Compatible with the PicoBlaze processors</td>
<td>PacoBlaze</td>
</tr>
</tbody>
</table>
μPabs

PC

IR

IRLoad

Reset

CS

ROM

A7-0

D7-0

256x8

sub

4

0

1

4

"0001" IR3-0

opfetch

clk

reset

Aeq

Control Unit

ALUSEl

Ase1

writeAcc

IRLoad

IR4-0

rbe

32x8 RegFile

OR

Oload

Reset

output

input

we

IR4-0

writeAcc

rbe

B

Acc

ALUSEl-0

Ase11-0

ALU

"000" & IR4-0

İzmir Institute of Technology

Computer Architecture
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;

entity datapath is port(
clk: in std_logic;
reset: in std_logic;
input: in std_logic_vector(7 downto 0);
output: out std_logic_vector(7 downto 0);
-- status signals
Aeq: out std_logic;
Jeq: out std_logic_vector(7 downto 0);
-- control signals
ALUSel: in std_logic_vector(1 downto 0);
Aeq: in std_logic;
writeAcc: in std_logic;
IRload: in std_logic;
PControl: in std_logic;
Oload: in std_logic;
jumpExec: in std_logic;
wr: in std_logic;
rbr: in std_logic);
end datapath;

architecture imp of datapath is

signal dp_ROMData, dp_IR, dp_IR2, dp_ALU_Out: std_logic_vector(7 downto 0);
signal dp_PC, dp_PCnext, dp_Adder_Out: std_logic_vector(7 downto 0);
signal dp_regfile_A, dp_regfile_B: std_logic_vector(7 downto 0);
signal dp_max4_Out: std_logic_vector(7 downto 0);
signal dp_max2_Out: std_logic_vector(3 downto 0);
signal dp_max2_Out8: std_logic_vector(7 downto 0);
signal f_unsigned_overflow: std_logic;
signal sub_jmp: std_logic;

begin
Aeq <= dp_regfile_A(0) or dp_regfile_A(1) or dp_regfile_A(2) or dp_regfile_A(3);  
or dp_regfile_A(4) or dp_regfile_A(5) or dp_regfile_A(6) or dp_regfile_A(7);

end;
Running the CPU
Pipelining