Overview

- Part 1 – Datapaths
  - Introduction
  - Datapath Example
  - Datapath Representation and Control Word
- Part 2 – A Simple Computer
  - Instruction Set Architecture (ISA)
  - Single-Cycle Hardwired Control
    - PC Function
    - Instruction decoder
    - Example instruction execution
- Part 3 – Multiple Cycle Hardwired Control
  - Single Cycle Computer Issues
  - Sequential Control Design

Instruction Set Architecture (ISA) for Simple Computer (SC)

- Instructions are stored in RAM or ROM as a program, the addresses for instructions are provided by a program counter (PC)
  - Count up or load a new address
  - The PC and associated control logic are part of the Control Unit
- A typical instruction specifies:
  - Operands to use
  - Operation to be performed
  - Where to place the result, or which instruction to execute next
- Executing an instruction
  - Activate the necessary sequence of operations specified by the instruction
  - Be controlled by the control unit and performed in:
    - datapath
    - control unit
    - external hardware such as memory or input/output

Example ISAs

- RISC (Reduced Instruction Set Computer)
  - Digital Alpha
  - Sun Sparc
  - MIPS R0000
  - IBM PowerPC
  - HP PA/RISC
- CISC (Complex Instruction Set Computer)
  - Intel x86
  - Motorola 68000
  - DEC VAX
- VLIW (Very Large Instruction Word)
  - Intel Itanium

ISA: Storage Resources

- "Harvard architecture": Separate instruction and data memories
- Permit use of single clock cycle per instruction implementation
- Due to use of "cache" in modern computer architectures, it is a fairly realistic model

Instruction memory $2^{16} \times 16$

Register file $8 \times 16$

Data memory $2^{16} \times 16$

Program counter (PC)
The three formats are: Register, Immediate, and Jump/Branch.

The Opcode specifies the operation to be performed:

\[ R1 \leftarrow R2 + R3 \]

The B Source Register field is replaced by an Operand field OP:

The SA field: permits jumps and branches on N or Z based on:

This format supports:

- Why is R2 in the second example SB instead of SA?

This instruction supports changes in the sequence of instruction execution by adding an extended, 6-bit, signed 2’s-complement address offset to the PC value:

The SA field: permits jumps and branches on N or Z based on the contents of Source register A:

The Address (AD) field (6-bit) replaces the DR and SB fields:

- Example: Suppose that a jump for the Opcode and the PC contains 45 (0...0101101) and AD contains –12 (1110100). Then the new PC value will be:

\[ 0...0101101 + (\ldots 1110100) = 0...0100001 \] (i.e., 45 + (–12) = 33)
Based on the ISA defined, the Data Memory has been attached to the Branch Control determines the PC transfers based on the followings:

- **PL**: Simply pass
- **JB**: Jump
- **RW**: Register Write

The architecture is to fetch and execute each instruction in a single clock cycle.

**Program Counter (PC) Function**

- PC function is based on instruction specifications involving jumps and branches:
  - Branch on Zero: **BRZ** if \( R(SA) = 0 \) \( \rightarrow \) PC ← PC + AD
  - Branch on Negative: **BRN** if \( R(SA) < 0 \) \( \rightarrow \) PC ← PC + AD
  - Jump: **JMP** \( \rightarrow \) PC ← R[SA]

The first two transfers require addition to the PC of:
- Address Offset = Extended IR(8:6) || IR(2:0)
- Jump Address = Bus A = R[SA]

In addition to the above register transfers, the PC must implement the counting function:
- PC ← PC + 1

**Instruction Decoder**

- Converts the instruction into the signals necessary to control the computer during the single cycle execution, combinational
  - Inputs: the 16-bit Instruction
  - Outputs: control signals
    - DA, AA, and BA: Register file addresses \( IR(8:0) \)
    - Simply pass-through signals: DA = DR, AA = SA, and BA = SB
    - FS: Function Unit Select
    - MB and MD: Multiplexer Select Controls
    - RW and MW: Register file and Data Memory Write Controls
    - PL, JB, and BC: PC Controls

Observe that for other than branches and jumps, FS = IR(12:9)

The other control signals should depend as much as possible on IR(15:13)

**The Control Unit**

- Datapath: the Data Memory has been attached to the Address Out, Data Out, and Data In lines of the Datapath.

- **Control Unit**:
  - The MW input to the Data Memory is the Memory Write signal from the Control Unit.
  - The Instruction Memory address input is provided by the PC and its instruction output feeds the Instruction Decoder.
  - Zero-filled IR(2:0) becomes Constant In
  - Extended IR(8:6) || IR(2:0) and Bus A are address inputs to the PC.
  - The PC is controlled by Branch Control logic

**PC Function (Contd.)**

- Branch Control determines the PC transfers based on five inputs:
  - **N.Z**: negative and zero status bits
  - **PL**: load enable for the PC
  - **JB**: Jump/Branch select: If JB = 1, Jump, else Branch
  - **BC**: Branch Condition select: If BC = 1, branch for N = 1, else branch for Z = 1.

<table>
<thead>
<tr>
<th>PL</th>
<th>JB</th>
<th>BC</th>
<th>PC Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>Count Up</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>X</td>
<td>Jump</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Branch on Negative (else Count Up)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Branch on Zero (else Count Up)</td>
</tr>
</tbody>
</table>

**Instruction Decoder (Contd.)**

<table>
<thead>
<tr>
<th>Instruction Function Type</th>
<th>Instruction Bits</th>
<th>Control Word Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Function unit operations using registers</td>
<td>0 0 0</td>
<td>0 0 1 0 0 X X</td>
</tr>
<tr>
<td>2. Memory read</td>
<td>0 0 1</td>
<td>X 1 1 0 0 X X</td>
</tr>
<tr>
<td>3. Memory write</td>
<td>0 1 0</td>
<td>0 X 0 1 0 X X</td>
</tr>
<tr>
<td>4. Function unit operations using register and constant</td>
<td>1 0 0</td>
<td>1 0 1 0 X X</td>
</tr>
<tr>
<td>5. Conditional branch on zero (Z)</td>
<td>1 1 0</td>
<td>X X 0 0 1 0 0</td>
</tr>
<tr>
<td>6. Conditional branch on negative</td>
<td>1 1 0</td>
<td>X X 0 0 1 0 1</td>
</tr>
<tr>
<td>Unconditional Jump</td>
<td>1 1 1</td>
<td>X X 0 0 1 1 X</td>
</tr>
</tbody>
</table>

**Truth Table for Instruction Decoder Logic**

- Table provides the necessary control signals for various instruction types.
- Each row represents a different instruction type with corresponding control word bits.

---

**Diagram and Notes**

- The diagram illustrates the flow of data and control signals in the computer architecture.
- Key components include Instruction Decoder, Instruction Memory, and Branch Control logic.
- Diagrams are used to visualize the hardware components and their interconnections.

---

**Additional Notes**

- The design philosophy is to enable efficient execution of instructions in a single cycle.
- The use of hardwired control ensures a predictable and fast execution path.
- Instruction decoding and control logic are crucial for translating high-level instructions into low-level operations.
- Diagrams serve as a visual aid to understand the flow and interaction of different parts of the computer system.
Instruction Decoder (Contd.)

- Instruction types are based on the control blocks and the seven control signals to be generated (MB, MD, RW, MW, PL, JB, BC):
  - Datapath and Memory Control (types 1-4)
    - Mux B
    - Memory and Mux D
  - PC Control (types 5-7)
    - Bit 15 = Bit 14 = 1 => PL
    - Bit 13 = JB.
    - Bit 9 was use as BC which contradicts FS = 0000 needed for branches. To force FS(0) to 0 for branches, Bit 9 into FS(0) is disabled by PL.

- The end result by use of the types, careful assignment of codes, and use of don't cares, yields very simple logic:
- This completes the design of most of the essential parts of the single-cycle simple computer.

Example Instruction Execution

<table>
<thead>
<tr>
<th>Operation code</th>
<th>Symbol or name</th>
<th>Description</th>
<th>Function</th>
<th>MB MD RW MW PL JB BC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000 010</td>
<td>ADI</td>
<td>Immediate</td>
<td>D[15:0]</td>
<td>1 0 1 0 0 0 0 0</td>
</tr>
<tr>
<td>0010 000</td>
<td>LD</td>
<td>Load mem.</td>
<td>Z[3:0]</td>
<td>0 1 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0100 000</td>
<td>ST</td>
<td>Store reg.</td>
<td>R[3:0]</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0001 010</td>
<td>SHL</td>
<td>Shift left</td>
<td>Z[3:0]</td>
<td>1 1 0 0 0 0 0 1</td>
</tr>
<tr>
<td>1100 000</td>
<td>BRZ</td>
<td>Jump/Branch</td>
<td>PC + R[3:0]</td>
<td>1 0 0 0 0 0 0 0</td>
</tr>
</tbody>
</table>

Decoding for ADI

- Decoding, control inputs and paths shown for ADI, LD and BRZ on next 6 slides

Decoding for LD