Overview

- Part 1 – Datapaths
  - Introduction
  - Datapath Example
  - Arithmetic Logic Unit (ALU)
  - Shifter
  - 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)

- A programmable system uses a sequence of *instructions* to control its operation.

- An typical instruction specifies:
  - Operation to be performed
  - Operands to use, and
  - Where to place the result, or
  - Which instruction to execute next

- Instructions are stored in RAM or ROM as a *program*.

- The addresses for instructions in a computer are provided by a *program counter (PC)* that can
  - Count up
  - Load a new address based on an instruction and, optionally, status information.
Instruction Set Architecture (ISA) (continued)

- The PC and associated control logic are part of the Control Unit
- Executing an instruction - activating the necessary sequence of operations specified by the instruction
- Execution is controlled by the control unit and performed:
  - In the datapath
  - In the control unit
  - In external hardware such as memory or input/output
ISA: Storage Resources

- The storage resources are "visible" to the programmer at the lowest software level (typically, machine or assembly language).
- Storage resources for the SC =>
  - Separate instruction and data memories imply "Harvard architecture"
  - Done to permit use of single clock cycle per instruction implementation
  - Due to use of "cache" in modern computer architectures, is a fairly realistic model

<table>
<thead>
<tr>
<th>Instruction memory (2^{15} \times 16)</th>
<th>Program counter (PC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data memory (2^{15} \times 16)</td>
<td>Register file (8 \times 16)</td>
</tr>
</tbody>
</table>
ISA: Instruction Format

- A instruction consists of a bit vector
- The fields of an instruction are subvectors representing specific functions and having specific binary codes defined
- The format of an instruction defines the subvectors and their function
- An ISA usually contains multiple formats
- The SC ISA contains the three formats presented on the next slide
ISA: Instruction Format

- The three formats are: Register, Immediate, and Jump and Branch.
- All formats contain an Opcode field in bits 9 through 15.
- The Opcode specifies the operation to be performed.
- More details on each format are provided on the next three slides.

<table>
<thead>
<tr>
<th>(a) Register</th>
<th>Opcode</th>
<th>Destination register (DR)</th>
<th>Source register A (SA)</th>
<th>Source register B (SB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>15 9 8 6 5 3 2 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(b) Immediate</th>
<th>Opcode</th>
<th>Destination register (DR)</th>
<th>Source register A (SA)</th>
<th>Operand (OP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>15 9 8 6 5 3 2 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(c) Jump and Branch</th>
<th>Opcode</th>
<th>Address (AD) (Left)</th>
<th>Source register A (SA)</th>
<th>Address (AD) (Right)</th>
</tr>
</thead>
<tbody>
<tr>
<td>15 9 8 6 5 3 2 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
ISA: Instruction Format (continued)

- This format supports instructions represented by:
  - $R1 \leftarrow R2 + R3$
  - $R1 \leftarrow \text{sl } R2$

- There are three 3-bit register fields:
  - DR - specifies destination register ($R1$ in the examples)
  - SA - specifies the A source register ($R2$ in the first example)
  - SB - specifies the B source register ($R3$ in the first example and $R2$ in the second example)

- Why is $R2$ in the second example SB instead of SA?
  - The source for the shifter in our datapath to be used in
This format supports instructions described by:

- R1 ← R2 + 3

The B Source Register field is replaced by an Operand field OP which specifies a constant.

The Operand:
- 3-bit constant
- Values from 0 to 7

The constant:
- Zero-fill (on the left of) the Operand to form 16-bit constant
  - 16-bit representation for values 0 through 7
ISA: Instruction Format (continued)

- This instruction supports changes in the sequence of instruction execution by adding an extended, 6-bit, signed 2s-complement address offset to the PC value.
- The 6-bit Address (AD) field replaces the DR and SB fields.
  - Example: Suppose that a jump is specified by the Opcode and the PC contains 45 (0…0101101) and Address contains –12 (110100). Then the new PC value will be:
    0…0101101 + (1…110100) = 0…0100001 (45 + (–12) = 33)
- The SA field is retained to permit jumps and branches on N or Z based on the contents of Source register A.
ISA: Instruction Specifications

- The specifications provide:
  - The name of the instruction
  - The instruction's opcode
  - A shorthand name for the opcode called a mnemonic
  - A specification for the instruction format
  - A register transfer description of the instruction, and
  - A listing of the status bits that are meaningful during an instruction's execution (not used in the architectures defined in this chapter)
### Instruction Specifications for the SimpleComputer - Part 1

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Opcode</th>
<th>Mnemonic</th>
<th>Format</th>
<th>Description</th>
<th>Status Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Increment</td>
<td>00000001</td>
<td>INC</td>
<td>RD,RA</td>
<td>R[DR] ← R[SA] + 1</td>
<td>N, Z</td>
</tr>
</tbody>
</table>
## ISA: Instruction Specifications (continued)

### Instruction Specifications for the Simple Computer - Part 2

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Opcode</th>
<th>Mnemonic</th>
<th>Format</th>
<th>Description</th>
<th>Status Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Move B</td>
<td>0001100</td>
<td>MOV B</td>
<td>RD,RB</td>
<td>R[DR] ← R[SB]</td>
<td></td>
</tr>
<tr>
<td>Shift Right</td>
<td>0001101</td>
<td>SHR</td>
<td>RD,RB</td>
<td>R[DR] ← sr R[SB]</td>
<td></td>
</tr>
<tr>
<td>Shift Left</td>
<td>0001110</td>
<td>SHL</td>
<td>RD,RB</td>
<td>R[DR] ← sl R[SB]</td>
<td></td>
</tr>
<tr>
<td>Load Immediate</td>
<td>1001100</td>
<td>LDI</td>
<td>RD, OP</td>
<td>R[DR] ← zf OP</td>
<td></td>
</tr>
<tr>
<td>Load</td>
<td>0010000</td>
<td>LD</td>
<td>RD,RA</td>
<td>R[DR] ← M[SA]</td>
<td></td>
</tr>
<tr>
<td>Store</td>
<td>0100000</td>
<td>ST</td>
<td>RA,RB</td>
<td>M[SA] ← R[SB]</td>
<td></td>
</tr>
<tr>
<td>Branch on Zero</td>
<td>1100000</td>
<td>BRZ</td>
<td>RA,AD</td>
<td>if (R[SA] = 0) PC ← PC + se AD</td>
<td></td>
</tr>
<tr>
<td>Branch on Negative</td>
<td>1100001</td>
<td>BRN</td>
<td>RA,AD</td>
<td>if (R[SA] &lt; 0) PC ← PC + se AD</td>
<td></td>
</tr>
<tr>
<td>Jump</td>
<td>1110000</td>
<td>JMP</td>
<td>RA</td>
<td>PC ← R[SA]</td>
<td></td>
</tr>
</tbody>
</table>
## ISA: Example Instructions and Data in Memory

### Memory Representation of Instructions and Data

<table>
<thead>
<tr>
<th>Decimal Address</th>
<th>Memory Contents</th>
<th>Decimal Opcode</th>
<th>Other Fields</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>0000101 001 010 011</td>
<td>5 (Subtract)</td>
<td>DR:1, SA:2, SB:3</td>
<td>R1 ← R2 − R3</td>
</tr>
<tr>
<td>35</td>
<td>0100000 000 100 101</td>
<td>32 (Store)</td>
<td>SA:4, SB:5</td>
<td>M[R4] ← R5</td>
</tr>
<tr>
<td>45</td>
<td>100010 010 111 011</td>
<td>66 (Add Immediate)</td>
<td>DR: 2, SA:7, OP:3</td>
<td>R2 ← R7 + 3</td>
</tr>
<tr>
<td>55</td>
<td>110000 101 110 100</td>
<td>96 (Branch on Zero)</td>
<td>AD: 44, SA:6</td>
<td>If R6 = 0, PC ← PC − 20</td>
</tr>
<tr>
<td>70</td>
<td>000000000011000000</td>
<td>Data = 192. After execution of instruction in 35, Data = 80.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

*Logic and Computer Design Fundamentals*  
*PowerPoint® Slides*  
© 2004 Pearson Education, Inc.
Single-Cycle Hardwired Control

- Based on the ISA defined, design a computer architecture to support the ISA
- The architecture is to fetch and execute each instruction in a single clock cycle
- The datapath from Figure 10-11 will be used
- The control unit will be defined as a part of the design
- The block diagram is shown on the next slide
The Control Unit

- The Data Memory has been attached to the Address Out and Data Out and Data In lines of the Datapath.
- The MW input to the Data Memory is the Memory Write signal from the Control Unit.
- For convenience, the Instruction Memory, which is not usually a part of the Control Unit is shown within it.
- 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

- PC function is based on instruction specifications involving jumps and branches taken from Slide 13:
  - Branch on Zero (BRZ) if (R[SA] = 0) PC ← PC + se AD
  - Branch on Negative (BRN) if (R[SA] < 0) PC ← PC + se AD
  - Jump (JMP) PC ← R[SA]

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

- The first two transfers above require addition to the PC of: Address Offset = Extended IR(8:6) || IR(2:0)

- The third transfer requires that the PC be loaded with: Jump Address = Bus A = R[SA]

- The counting function of the PC requires addition to the PC of 1
PC Function (continued)

- Branch Control determines the PC transfers based on five of its inputs defined as follows:
  - 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.

- The above is summarize by the following table:

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

- Sufficient information is provided here to design the PC
Instruction Decoder

- The combinational instruction decoder converts the instruction into the signals necessary to control all parts of the computer during the single cycle execution.
- The input is the 16-bit Instruction.
- The outputs are control signals:
  - Register file addresses DA, AA, and BA,
  - Function Unit Select FS,
  - Multiplexer Select Controls MB and MD,
  - Register file and Data Memory Write Controls RW and MW, and
  - PC Controls PL, JB, and BC.
- The register file outputs are simply pass-through signals: DA = DR, AA = SA, and BA = SB.
- Determination of the remaining signals is more complex.
Instruction Decoder (continued)

- The remaining control signals do not depend on the addresses, so must be a function of IR(13:9)
- Formulation requires examining relationships between the outputs and the opcodes given in Slides 12 and 13.
- Observe that for other than branches and jumps, FS = IR(12:9)
- This implies that the other control signals should depend as much as possible on IR(15:13) (which actually were assigned with decoding in mind!)
- To make some sense of this, we divide instructions into types as shown in the table on the next page
# Instruction Decoder (continued)

## Truth Table for Instruction Decoder Logic

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

Instruction Function Type:
- 15 14 13 9

Control Word Bits:
- MB
- MD
- RW
- MW
- PL
- JB
- BC
Instruction Decoder (continued)

- The types are based on the blocks controlled and the seven signals to be generated; types can be divided into two groups:
  - Datapath and Memory Control (First 4 types)
  - PC Control (Last 3 types)
- In Datapath and Memory Control blocks controlled are considered:
  - Mux B (1st and 4th types)
  - Memory and Mux D (2nd and 3rd types)
  - By assigning codes with no or only one 1 for these, implementation of MB, MD, RW and MW are simplified.
- In Control Unit more of a bit setting approach was used:
  - Bit 15 = Bit 14 = 1 were assigned to generate PL
  - Bit 13 values were assigned to generate JB.
  - Bit 9 was use as BC which contradicts FS = 0000 needed for branches. To force FS(6) to 0 for branches, Bit 9 into FS(6) is disabled by PL.
- Also, useful bit correlations between values in the two groups were exploited in assigning the codes.
Instruction Decoder (continued)

- 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

#### Six Instructions for the Single-Cycle Computer

<table>
<thead>
<tr>
<th>Operation code</th>
<th>Symbolic name</th>
<th>Format</th>
<th>Description</th>
<th>Function</th>
<th>MB</th>
<th>MD</th>
<th>RW</th>
<th>MW</th>
<th>PL</th>
<th>JB</th>
<th>BC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000010</td>
<td>ADI</td>
<td>Immediate</td>
<td>Add immediate operand</td>
<td>$R[DR] \leftarrow R[SA] + zf (2:0)$</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0010000</td>
<td>LD</td>
<td>Register</td>
<td>Load memory content into register</td>
<td>$R[DR] \leftarrow M[R[SA]]$</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0100000</td>
<td>ST</td>
<td>Register</td>
<td>Store register content in memory</td>
<td>$M[R[SA]] \leftarrow R[SB]$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0001110</td>
<td>SL</td>
<td>Register</td>
<td>Shift left</td>
<td>$R[DR] \leftarrow sl R[SB]$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0001011</td>
<td>NOT</td>
<td>Register</td>
<td>Complement register</td>
<td>$R[DR] \leftarrow \overline{R[SA]}$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1100000</td>
<td>BRZ</td>
<td>Jump/Branch</td>
<td>If $R[SA] = 0$, branch to PC + se AD</td>
<td>If $R[SA] = 0$, $PC \leftarrow PC + se AD$, If $R[SA] \neq 0$, $PC \leftarrow PC + 1$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

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

![Instruction and Control Word Diagram]

**Instruction**

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8–6</th>
<th>5–3</th>
<th>2–0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Control word**

- 19-17: DA
- 16-14: AA
- 13-11: BA
- 10: MB
- 9-6: FS
- 5: MD
- 4: RW
- 3: MW
- 2: PL
- 1: JB
- 0: BC

The diagram illustrates the decoding process for ADI, showing how the instruction is parsed into various components for further processing.
Decoding for LD

Instruction

<table>
<thead>
<tr>
<th>Opcode</th>
<th>DR</th>
<th>SA</th>
<th>SB</th>
</tr>
</thead>
<tbody>
<tr>
<td>0010 0000</td>
<td>15 14 13 12 11 10 9</td>
<td>8 6</td>
<td>5 3</td>
</tr>
</tbody>
</table>

Control word

19-17 16-14 13-11 10 9-6 5-3 4 3 2 1 0
DA AA BA MB FS MD RW MW PL JB BC

Chapter 10  Part 2  28
Decoding for **BRZ**

![Diagram of decoding for BRZ]

**Instruction**

```
1100000
```

**Control word**

```
1917 16-14 13-11 10 9-6 5 4 3 2 1 0 0 0 0 0 0 0
```

- **DA**
- **AA**
- **BA**
- **MB**
- **FS**
- **MD**
- **RW**
- **MW**
- **PL**
- **JB**
- **BC**
Control Inputs and Paths for BRZ

- **Branch on Z**: If Z is 1, branch on zero.
- **Zero fill**: IR(2:0) to zero fill.
- **Address out**: From IR(8:6) || IR(2:0).
- **No Write**: On Z.
- **Instruction memory**: For BRZ.
- **Register file**: A → B → BA.
- **Constant in**: To function unit.
- **Function unit**: MB → 0000 → 0001 → 0010 → 0100.
- **Data memory**: Data in Address.
- **Data out**: No Write.
- **Data in Address**: Data out.
Terms of Use

- © 2004 by Pearson Education, Inc. All rights reserved.
- The following terms of use apply in addition to the standard Pearson Education Legal Notice.
- Permission is given to incorporate these materials into classroom presentations and handouts only to instructors adopting Logic and Computer Design Fundamentals as the course text.
- Permission is granted to the instructors adopting the book to post these materials on a protected website or protected ftp site in original or modified form. All other website or ftp postings, including those offering the materials for a fee, are prohibited.
- You may not remove or in any way alter this Terms of Use notice or any trademark, copyright, or other proprietary notice, including the copyright watermark on each slide.
- Return to Title Page