Section 5

Basic Computer Organization and Design

Mano's Basic Computer

- Memory unit with 4096 16-bit words
- Registers: \( AR, PC, DR, AC, IR, TR, OUTR, INPR, SC \)
- Flip-flops: \( I, S, E, R, IEN, FGI, FGO \)
- 3 x 8 op decoder and 4 x 16 timing decoder
- 16-bit common bus
- Control logic gates
- Adder and logic circuit connected to input of \( AC \)

Instruction Code

- Computer instruction is binary code that specifies a sequence of microoperations
- Operation code + Address
  - Op code must have \( n \) bits for \( \leq 2^n \) operations
  - Op code sometimes called a macrooperation
  - Address is register or memory location
    - Memory location is operand address
- Shorten “instruction code” to “instruction”
- Instructions and data in memory

Stored Program Organization

- One processor register
  - \( AC \) – accumulator
- Instruction format
  - 4-bit op code
  - 12-bit address (for \( 2^{12} = 4096 \) memory words)
- Instruction execution cycle
  - Read 16-bit instruction from memory
  - Use 12-bit address to fetch operand from memory
  - Execute 4-bit op code

Address Types

- 12-bit instruction address
  - Immediate
    - Actual data value
  - Direct
    - Memory address where data (operand) resides
  - Indirect
    - Memory address where memory address of data (operand) resides
  - Effective address is the address of the operand
  - Lead bit of instruction used as indirect flag

© 2004 by Ted Borys. All rights reserved.
Direct / Indirect Address

Program Counter (PC)
- Holds memory address of next instruction
- Next instruction is fetched after current instruction completes execution cycle
- PC is incremented right after instruction is fetched from memory
- PC value can be replaced by new address when executing a branch instruction

Register Control Inputs
- Load (LD)
- Increment (INR)
- Clear (CLR)

Basic Computer Registers

<table>
<thead>
<tr>
<th>Register</th>
<th>Number of bits</th>
<th>Register name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>DR</td>
<td>16</td>
<td>Data register</td>
<td>Holds memory operand</td>
</tr>
<tr>
<td>AR</td>
<td>12</td>
<td>Address register</td>
<td>Holds address for memory</td>
</tr>
<tr>
<td>AC</td>
<td>16</td>
<td>Accumulator</td>
<td>Processor register</td>
</tr>
<tr>
<td>IR</td>
<td>16</td>
<td>Instruction register</td>
<td>Holds instruction code</td>
</tr>
<tr>
<td>PC</td>
<td>12</td>
<td>Program counter</td>
<td>Holds address of instruction</td>
</tr>
<tr>
<td>TR</td>
<td>16</td>
<td>Temporary register</td>
<td>Holds temporary data</td>
</tr>
<tr>
<td>INPR</td>
<td>8</td>
<td>Input register</td>
<td>Holds input character</td>
</tr>
<tr>
<td>OUTR</td>
<td>8</td>
<td>Output register</td>
<td>Holds output character</td>
</tr>
</tbody>
</table>

Registers + Memory Layout

Common Bus
- Connects registers and memory
- Specific output selected by \( S_2S_1S_0 \)
  - When register has < 16 bits, high-order bus bits are set to 0
- Register with LD enabled reads data from bus
- Memory with Write enabled reads bus
- Memory with Read enabled puts data on bus
  - When \( S_2S_1S_0 = 111 \)
Address Register (AR)

- Always used to specify address within memory unit
- Dedicated register eliminates need for separate address bus
- Content of any register output connected to the bus can be written to memory
- Any register input connected to bus can be target of memory read
  - As long as its LD is enabled

Accumulator (AC)

- Input comes from adder and logic circuit
- Adder and logic circuit
  - Input
    - 16-bit output of AC
    - 16-bit data register (DR)
    - 8-bit input register (INPR)
  - Output
    - 16-bit input of AC
    - E flip-flop (extended AC bit, aka overflow)
- DR and AC input used for arithmetic and logic microoperations

Timing Is Everything

- Content of any register output connected to the bus can be applied to the bus and content of any register input connected to the bus can be loaded from the bus during the same clock cycle
- These 2 microoperations can be executed at the same time
  \[ DR \leftarrow AC \text{ and } AC \leftarrow DR \]

Bus Connections

Basic Instruction Formats

Instruction Format

- Only 3 bits used for op code
- Looks like only 8 different op codes are possible
- Wrong!
- For op code 111, one of the low-order 12 bits is turned on to extend the op code definition
Basic Instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>0000</td>
<td>Add one memory word to AC</td>
</tr>
<tr>
<td>SUB</td>
<td>0001</td>
<td>Subtract one memory word to AC</td>
</tr>
<tr>
<td>MUL</td>
<td>0010</td>
<td>Multiply one memory word to AC</td>
</tr>
<tr>
<td>DIV</td>
<td>0011</td>
<td>Divide one memory word to AC</td>
</tr>
<tr>
<td>BHR</td>
<td>0100</td>
<td>Branch unconditionally</td>
</tr>
<tr>
<td>BRN</td>
<td>0101</td>
<td>Branch to memory address</td>
</tr>
<tr>
<td>BLS</td>
<td>0110</td>
<td>Branch on less than or equal to</td>
</tr>
<tr>
<td>CLI</td>
<td>0111</td>
<td>Clear AC</td>
</tr>
<tr>
<td>OUT</td>
<td>1000</td>
<td>Output character from AC</td>
</tr>
<tr>
<td>IN</td>
<td>1001</td>
<td>Input character to AC</td>
</tr>
<tr>
<td>STG</td>
<td>1010</td>
<td>Store character to AC</td>
</tr>
<tr>
<td>DCH</td>
<td>1011</td>
<td>Clear character</td>
</tr>
<tr>
<td>SCL</td>
<td>1100</td>
<td>Set clear flag</td>
</tr>
<tr>
<td>SFR</td>
<td>1101</td>
<td>Set flag for register</td>
</tr>
<tr>
<td>SFR</td>
<td>1110</td>
<td>Set flag for register</td>
</tr>
<tr>
<td>SFR</td>
<td>1111</td>
<td>Set flag for register</td>
</tr>
</tbody>
</table>

Instruction Set Completeness

- Arithmetic, logical, and shift
- Move data from and to memory and registers
- Program control and status check
- Input and output
  - (I/O, I/O, it’s off to the bus we go…)

Control Unit

- Instruction read from memory and put in IR
- Leftmost bit put in I flip-flop
- 3-bit op code decoded with 3 x 8 decoder into D0 to D7
- 4-bit sequence counter (SC) decoded with 4 x 16 decoder into T0 to T15 (timing signals)
- I, D0 to D7, T0 to T15, rightmost 12 bits of IR, and other inputs are fed into control and logic gates

Sequence Counter (SC)

- Inputs are increment (INR) and clear (CLR)
- Example
  - SC incremented to provide T0, T1, T2, T3, and T4
  - At time T4, SC is cleared to 0 if D3 is active
  - Written as: D4 T4: SC ← 0

Timing Diagram
### Instruction Cycle
- Fetch instruction from memory
- Decode the instruction
- Read effective address from memory if indirect address
- Execute the instruction

### Fetch And Decode
- SC cleared to 0, generating timing signal $T_0$
- After each clock pulse, SC is incremented
- Fetch and decode microoperations
  - $T_0$: $AR \leftarrow PC$
  - $T_1$: $IR \leftarrow M[AR], PC \leftarrow PC + 1$
  - $T_2$: $D_0…D_7 \leftarrow$ decode $IR(12-14)$, $AR \leftarrow IR(0-11), I \leftarrow IR(15)$

### Fetch Phase

### Instruction Cycle Flowchart

### Instruction Paths
- $D_7T_5$: $AR \leftarrow M[AR]$
- $D_7T_5$: Do nothing
- $D_7T_5$: Execute a register-reference instruction
- $D_7T_5$: Execute an I/O instruction

### Register-Reference Instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r$:</td>
<td>SC ← 0</td>
</tr>
<tr>
<td>CLA</td>
<td>Clear SC</td>
</tr>
<tr>
<td>CLE</td>
<td>Clear E</td>
</tr>
<tr>
<td>CMA</td>
<td>$AC \leftarrow \overline{AC}$</td>
</tr>
<tr>
<td>CME</td>
<td>Complement AC</td>
</tr>
<tr>
<td>CIR</td>
<td>$AC \leftarrow \text{shl AC}, AC(15) \leftarrow E, E \leftarrow AC(0)$</td>
</tr>
<tr>
<td>CIL</td>
<td>Circulate right</td>
</tr>
<tr>
<td>INC</td>
<td>$AC \leftarrow AC + 1$</td>
</tr>
<tr>
<td>SPA</td>
<td>$AC \leftarrow \overline{AC}$</td>
</tr>
<tr>
<td>SNA</td>
<td>Skip if positive</td>
</tr>
<tr>
<td>SZA</td>
<td>If $(AC = 0)$ then $(PC \leftarrow PC + 1)$</td>
</tr>
<tr>
<td>SZE</td>
<td>Skip if $E$ zero</td>
</tr>
<tr>
<td>HLT</td>
<td>$S \leftarrow 0$ (S is a start-stop flip-flop)</td>
</tr>
</tbody>
</table>

© 2004 by Ted Borys. All rights reserved.
Memory-Reference Instructions

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Operation decoder</th>
<th>Symbolic description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>$D_0$</td>
<td>$AC \leftarrow AC \land M[AR]$</td>
</tr>
<tr>
<td>ADD</td>
<td>$D_1$</td>
<td>$AC \leftarrow AC + M[AR], \ E \leftarrow C_{out}$</td>
</tr>
<tr>
<td>LDA</td>
<td>$D_2$</td>
<td>$AC \leftarrow M[AR]$</td>
</tr>
<tr>
<td>STA</td>
<td>$D_3$</td>
<td>$M[AR] \leftarrow AC$</td>
</tr>
<tr>
<td>BUN</td>
<td>$D_4$</td>
<td>$PC \leftarrow AR$</td>
</tr>
<tr>
<td>BSA</td>
<td>$D_5$</td>
<td>$M[AR] \leftarrow PC, \ PC \leftarrow AR + 1$</td>
</tr>
<tr>
<td>ISZ</td>
<td>$D_6$</td>
<td>$M[AR] \leftarrow M[AR] + 1, \ $ \text{If } M[AR] + 1 = 0 \text{ then } PC \leftarrow PC + 1$</td>
</tr>
</tbody>
</table>

AND to AC

- $D_0T_4$: $DR \leftarrow M[AR]$
- $D_0T_5$: $AC \leftarrow AC \land DR, \ SC \leftarrow 0$

ADD to AC

- $D_1T_4$: $DR \leftarrow M[AR]$
- $D_1T_5$: $AC \leftarrow AC + DR, \ E \leftarrow C_{out}, \ SC \leftarrow 0$

LDA: Load AC

- $D_2T_4$: $DR \leftarrow M[AR]$
- $D_2T_5$: $AC \leftarrow DR, \ SC \leftarrow 0$

STA: Store AC

- $D_3T_4$: $M[AR] \leftarrow AC, \ SC \leftarrow 0$

BUN: Branch Unconditionally

- $D_4T_4$: $PC \leftarrow AR, \ SC \leftarrow 0$
BSA: Branch & Save Return Address

- $D_3T_4$: $M[AR] \leftarrow PC$, $AR \leftarrow AR + 1$
- $D_3T_5$: $PC \leftarrow AR$, $SC \leftarrow 0$

BSA Example

(ISZ: Increment & Skip if Zero)

- Increment word specified by effective address
  - If value = 0, increment $PC$
- $D_6T_4$: $DR \leftarrow M[AR]$
- $D_6T_5$: $DR \leftarrow DR + 1$
- $D_6T_6$: $M[AR] \leftarrow DR$, $SC \leftarrow 0$
  - if ($DR = 0$) then ($PC \leftarrow PC + 1$)

Memory-Reference Instructions

Input Register $INPR$

- 1-bit input flip-flop $FGI$
  - Initially cleared to 0
- When key hit on keyboard
  - 8-bit alphanumeric code is shifted into $INPR$
  - Input flag $FGI$ set to 1
  - No more input can be accepted from keyboard
- Computer checks $FGI$, when set to 1
  - Parallel transfer from $INPR$ to $AC$
  - $FGI$ cleared to 0
  - More input can now be accepted from keyboard

Output Register $OUTR$

- 1-bit output flip-flop $FGO$
  - Initially set to 1
- Computer checks $FGO$, when set to 1
  - Parallel transfer from $AC$ to $OUTR$
  - $FGO$ cleared to 0
  - No more output can be sent from computer
- Output device accepts 8-bit character
  - $FGO$ set to 1
  - More output can now be sent from computer
Interrupt Enable \textit{IEN}

- Having computer constantly check \textit{FGI} and \textit{FGO} via an executable instruction is a waste of time
- Instead, \textit{IEN} is programmatically set, effectively saying “let me know if you need me”
  - Meanwhile, it keeps executing instructions
- During each execution cycle, if computer detects \textit{FGI} or \textit{FGO} is set, then \textit{R} is set to 1
- The interrupt happens when the computer is ready to fetch the next instruction
  - \textit{R} = 0 means go through instruction cycle
  - \textit{R} = 1 means go through interrupt cycle

### Interrupt Cycle Example

- Condition for setting \textit{R} to 1:
  \[ T_0 \cdot T_1 \cdot T_2 (\text{IEN})(\text{FGI} + \text{FGO}): R \leftarrow 1 \]
- Fetch phase modified to service interrupt:
  \[
  RT_0: AR \leftarrow 0, \ TR \leftarrow PC \\
  RT_1: M[AR] \leftarrow TR, \ PC \leftarrow 0 \\
  RT_2: PC \leftarrow PC + 1, \ IEN \leftarrow 0, \ R \leftarrow 0, \ SC \leftarrow 0
  \]
Computer Operation Flowchart

Inputs To Control Logic Gates
- Two decoders
  - 8-bit instruction and 16-bit sequence
- Seven flip-flops: I, S, E, R, IEN, FGI, FGO
- IR bits 0 through 11
- AC bits 0 through 15
  - Check if $AC = 0$ and check sign bit
- DR bits 0 through 15
  - Check if $DR = 0$

Outputs Of Control Logic Gates
- Control inputs of nine registers
- Control read & write inputs of memory
- Set, clear, or complement flip-flops
- $S_2, S_1, S_0$ to select a register for the bus
- Control AC adder and logic circuit

AR Control Gates
- Register control inputs: LD, INR, and CLR
- Find all statements that alter $AR$ contents
  - $R'T_0$: $AR \leftarrow PC$ LD
  - $R'T_2$: $AR \leftarrow IR(0-11)$ LD
  - $D_7'T_3$: $AR \leftarrow M[AR]$ LD
  - $RT_0$: $AR \leftarrow 0$ CLR
  - $D_5'T_4$: $AR \leftarrow AR + 1$ INR

IEN Control Gates
- Find all statements that change $IEN$
  - $D_7'T_3B_7$: $IEN \leftarrow 1$
  - $D_7'T_3B_6$: $IEN \leftarrow 0$
  - $RT_2$: $IEN \leftarrow 1$
IEN Control Gates

Encoder For Bus Selection Circuit

Boolean Functions For Encoder

Boolean Function For $x_1$

Circuits Associated With AC

AC Control Gates
Mano’s Basic Computer

- Memory unit with 4096 16-bit words
- Registers: AR, PC, DR, AC, IR, TR, OUTR, INPR, SC
- Flip-flops: I, S, E, R, IEN, FGI, FGO
- 3 x 8 op decoder and 4 x 16 timing decoder
- 16-bit common bus
- Control logic gates
- Adder and logic circuit connected to input of AC