CS/EE 260. Digital Computers
Organization and Logical Design

David M. Zar
Computer Science and Engineering Department
Washington University

dzar@cse.wustl.edu
http://www.cse.wustl.edu/~dzar/class/260
Digital Computers and Information

- Digital computers
- Number representations
- Arithmetic operations
- Alphanumeric codes
What's in a Computer?

Microprocessor
- Central Processing Unit
- Floating Point Unit
- Memory Management Unit
- Internal Cache Memory

Main Memory
- stores running programs and associated data
- typically 8-64 Mbytes
- Dynamic RAM
What's in a Computer?

- Monitor
- Graphics Card
- Network Interface
- Disk Controller
- Peripheral Bus Interface
- Main Memory
- External Cache
- Memory Bus
- Control movement of data between memory and I/O devices
- Monitor
- Microprocessor
- FPU
- MMU
- CPU
- Internal Cache
- Control movement of data between memory and I/O devices
- Keyboard
- External Cache
- Main Memory
- Small, fast memory
- Stores recently used data
What's in a Computer?

- microprocessor
  - FPU
  - MMU
  - CPU
  - Internal cache
- Graphics Card
- Peripheral Bus Interface
- Network Interface
- Disk Controller
- Main Memory
- Peripheral Bus
- Monitor
- Keyboard
- Graphics Card
- Disk Controller
- Network Interface
- Peripheral Bus Interface
- External Cache

- Convert text & graphics to video
- Transfer data between external network & memory
- Read/write data on proper physical location on disk
- Transfer data between external network & memory
What's in a Chip?

Intel 80286 Microprocessor (www.micro.magnet.fsu.edu/chipshots)

NEC/MIPS R4400 Microprocessor (www.micro.magnet.fsu.edu/chipshots)

Photomicrographs made using colored filters.
What’s in this Chip?

- Component from WU gigabit ATM switch.
- Dense areas are Static RAM blocks.
- Other blocks implement various logic functions.
- Selected statistics.
  - .7 \( \mu \)m CMOS process
  - 1.4 \( \times \) 1.4 cm die
  - 1.4 million transistors
**Basic Processor & Memory**

- Memory stores programs and data.
  - organized as set of numbered storage slots
  - each storage slot (memory word) can hold a number
  - processor can read from or write to any word

- Fetch & execute cycle
  - read word whose address is in Program Counter (PC) and increment PC
  - interpret stored value as instruction (decoding)
  - perform instruction using Accumulator (ACC) and Arithmetic & Logic Unit (ALU)
Simple Instruction Set

0xx load the accumulator with value stored in memory word xx
1xx store the value in the accumulator into memory word xx
2xx add the value in memory word xx to the value in the accumulator
300 negate the value in the accumulator
301 halt
4xx change the value of the PC to xx
5xx if the value in the accumulator is zero, change PC value to xx
6xx load the accumulator with value whose address is stored in word xx
7xx store the accumulator value into the word whose address is in word xx
8xx change the accumulator value to xx
9xx add xx to the value in the accumulator
**Simple Program**

Add the values in locations 0-9 and write sum in 10.

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>10</td>
<td>Store sum here</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>Pointer to “next” value here</td>
</tr>
<tr>
<td>12 (start)</td>
<td>800 (load “00”)</td>
<td>initialize sum</td>
</tr>
<tr>
<td>13</td>
<td>110 (store 10)</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>111 (store 11)</td>
<td>initialize pointer</td>
</tr>
<tr>
<td>15 (loop)</td>
<td>810 (load “10”)</td>
<td>if pointer=10, then quit</td>
</tr>
<tr>
<td>16</td>
<td>300 (negate)</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>211 (add 11)</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>526 (if 0 goto 26)</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>611 (load *11)</td>
<td>sum = sum + *pointer</td>
</tr>
<tr>
<td>20</td>
<td>210 (add 10)</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>110 (store 10)</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>801 (load “1”)</td>
<td>pointer = pointer + 1</td>
</tr>
<tr>
<td>23</td>
<td>211 (add 11)</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>111 (store 11)</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>415 (goto 15)</td>
<td></td>
</tr>
<tr>
<td>26 (end)</td>
<td>301 (halt)</td>
<td></td>
</tr>
</tbody>
</table>
Representing Information in Computers

- Electronic computers represent information as voltage levels.
- To make the computer hardware simple and reliable, computers represent information in binary form.
  » example: voltages greater than 3V are interpreted as representing one value (called “1”), voltages less than 2V are interpreted as representing another value (called “0”).
- In principle, could use more voltage levels.
  » example: 0 to .75V represents “0”, 1 to 1.75V represents “1”, 2 to 2.75V represents “2”, and so forth.
- In practice, this is rarely done.
  » requires more complex circuits
  » circuits are more susceptible to noise, hence less reliable
Computers, like all electronic systems, are affected by noise.

- noise has various sources (nearby signal changes, thermal vibrations of molecules in semiconductor materials, ...)
- in computers, noise can cause binary signals to be misinterpreted

The noise margin is the amount of noise that a system can tolerate and still correctly identify a logic high or low.
Number Representation

- Standard decimal number representation
  
  \[ 243.83 = 2 \times 10^2 + 4 \times 10^1 + 3 \times 10^0 + 8 \times 10^{-1} + 3 \times 10^{-2} \]

- Generalization to base \( r \)
  
  \[ A_n \ldots A_1 A_0 A_{-1} \ldots A_{-m} \]
  
  \[ = A_n \times r^n + \ldots + A_1 \times r^1 + A_0 \times r^0 + A_{-1} \times r^{-1} + \ldots + A_{-m} \times r^{-m} \]

- Binary number representation
  
  \[ 110.01 = 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} \]

- Converting binary numbers to decimal (easy).
  
  write binary expansion, replace powers of 2 with decimal values, and add up the values
  
  \[ 110.01 = 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} = 4 + 2 + 0 + 0 + 1/4 = 6.25 \]

Note: it helps to know your powers of 2 (hint)
Decimal to Binary Conversion

- Repeated subtraction of powers of 2
  
  \[
  625 = 512 + 113 \quad 512 = 2^9 \\
  113 = 64 + 49 \quad 64 = 2^6 \\
  49 = 32 + 17 \quad 32 = 2^5 \\
  17 = 16 + 1 \quad 16 = 2^4 \\
  1 = 1 + 0 \quad 1 = 2^0
  \]

  So, \((625)_{10} = 2^9 + 2^6 + 2^5 + 2^4 + 2^0 = (1001110001)_2\)

- General procedure for integers.
  
  » Let \(v\) = value to be converted. Clear all output bits.
  » Repeat until \(v = 0\).
    - Let \(2^i\) be largest power of 2 that is \(\leq v\)
    - Set bit \(i\) of the output
    - Subtract \(2^i\) from \(v\)
Decimal-Binary Conversion (Method 2)

- Repeated division by 2
  
  \[
  \begin{align*}
  625/2 &= 312 \text{ with remainder of } 1 \quad \text{least significant bit} \\
  312/2 &= 156 \text{ with remainder of } 0 \\
  156/2 &= 78 \text{ with remainder of } 0 \\
  78/2 &= 39 \text{ with remainder of } 0 \\
  39/2 &= 19 \text{ with remainder of } 1 \\
  19/2 &= 9 \text{ with remainder of } 1 \\
  9/2 &= 4 \text{ with remainder of } 1 \\
  4/2 &= 2 \text{ with remainder of } 0 \\
  2/2 &= 1 \text{ with remainder of } 0 \\
  1/2 &= 0 \text{ with remainder of } 1 \\
  \\
  \text{So, } (625)_{10} &= (10\,0111\,0001)_2
  \end{align*}
\]
Octal and Hexadecimal

Octal (base 8) and hexadecimal (base 16) provide more convenient way for people to write binary numbers.

- \(110101100010 = 110\ 101\ 100\ 010 = (6542)_8\)
- \(1101\ 0110\ 0010\ = (d62)_{16}\)

<table>
<thead>
<tr>
<th>Octal Conversion</th>
<th>Hexadecimal Conversion</th>
</tr>
</thead>
<tbody>
<tr>
<td>000 = 0</td>
<td>0000 = 0 1000 = 8</td>
</tr>
<tr>
<td>001 = 1</td>
<td>0001 = 1 1001 = 9</td>
</tr>
<tr>
<td>010 = 2</td>
<td>0010 = 2 1010 = 10 = a</td>
</tr>
<tr>
<td>011 = 3</td>
<td>0011 = 3 1011 = 11 = b</td>
</tr>
<tr>
<td>100 = 4</td>
<td>0100 = 4 1100 = 12 = c</td>
</tr>
<tr>
<td>101 = 5</td>
<td>0101 = 5 1101 = 13 = d</td>
</tr>
<tr>
<td>110 = 6</td>
<td>0110 = 6 1110 = 14 = e</td>
</tr>
<tr>
<td>111 = 7</td>
<td>0111 = 7 1111 = 15 = f</td>
</tr>
</tbody>
</table>
Finite Data Representations

- Computer hardware is generally designed to operate on words with a fixed number of bits (e.g. 16 bits).
- Places a limit on the number of discrete values that can be stored in a single word (e.g. $2^{16}$).
- If we use words to represent positive integers then with $n$ bits, we can represent integers 0 up to $2^n - 1$.
- Larger integers can be represented by multiple words.
  - computer hardware operates on single words
  - software must combine results from single word operations to produce desired result
- Or, use floating point representation for large (and small) values; typically supported by computer hardware.
How Computers Add

- Binary long addition similar to decimal long addition.

<table>
<thead>
<tr>
<th></th>
<th>decimal</th>
<th>binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>carry</td>
<td>1100</td>
<td>111100</td>
</tr>
<tr>
<td>augend</td>
<td>2565</td>
<td>101110</td>
</tr>
<tr>
<td>addend</td>
<td>6754</td>
<td>110111</td>
</tr>
<tr>
<td>sum</td>
<td>9319</td>
<td>110001</td>
</tr>
</tbody>
</table>

- Binary addition algorithm - add $a_{n-1}...a_0$ to $b_{n-1}...b_0$ and put result in $s_n...s_0$

  $c_0=0$  // $c_i$ are carry bits

  for $i = 0$ to $n-1$

  if one or three of $a_i$, $b_i$ or $c_i$ are $= 1$ then $s_i = 1$ else $s_i = 0$

  if at least two of $a_i$, $b_i$ or $c_i$ are $= 1$, then $c_{i+1} = 1$ else $c_{i+1} = 0$

  $s_n = c_n$
Modular and Signed Arithmetic

- Computers use modular arithmetic in which values wrap around circularly.
  - to add \( A + B \), start at position for \( A \) and then count clockwise \( B \) positions
  - modular arithmetic is just like “clock arithmetic”

- Associating certain bit patterns with negative values yields signed arithmetic.

- Negate a given value by flipping all bits and adding 1.
- Must pay attention to overflow.
Representing Text

- Computers use numbers to represent alphabetic characters, numerals and punctuation.
- Most common encoding is ASCII (American Standard Code for Communication Interchange)
  - characters represented by 7 bit values
  - numerals start at \((30)_{16}\)
  - upper case letters start at \((41)_{16}\)
  - lower case letters start at \((61)_{16}\)
  - see Table 1-4 in Mano for details
- Unicode uses 16 bits per character, allowing it to represent far more distinct characters.
## Convert Numeric String to Internal Value

ASCII character codes for decimal integer stored in locations 0..3 with M.S.D. at location 0. Write internal value in location 4. Assume 16 bit words.

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0004</td>
<td>0004</td>
<td>Store result here</td>
</tr>
<tr>
<td>0005</td>
<td>0005</td>
<td>Pointer to “next” character</td>
</tr>
<tr>
<td>0006</td>
<td>0006</td>
<td>Store temporary value here</td>
</tr>
<tr>
<td>0007</td>
<td>8 000</td>
<td>(load “00”) result = 0</td>
</tr>
<tr>
<td>0008</td>
<td>1 004</td>
<td>(store 04)</td>
</tr>
<tr>
<td>0009</td>
<td>1 005</td>
<td>(store 05) pointer = 0</td>
</tr>
<tr>
<td>000a</td>
<td>8 004</td>
<td>(load “04”) if pointer = 4, then quit</td>
</tr>
<tr>
<td>000b</td>
<td>3 000</td>
<td>(negate)</td>
</tr>
<tr>
<td>000c</td>
<td>2 005</td>
<td>(add 05)</td>
</tr>
<tr>
<td>000d</td>
<td>5 01b</td>
<td>(if 0 goto 1b)</td>
</tr>
<tr>
<td>000e</td>
<td>8 00a</td>
<td>(load “0a”) result = 10 * result</td>
</tr>
<tr>
<td>000f</td>
<td>a 004</td>
<td>(mult 04) New multiply instruction</td>
</tr>
<tr>
<td>0010</td>
<td>1 004</td>
<td>(store 04)</td>
</tr>
</tbody>
</table>
# Convert Numeric String (continued)

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0011</td>
<td>8 fd0</td>
<td>(load “-48”) result = result</td>
</tr>
<tr>
<td>0012</td>
<td>1 006</td>
<td>(store 06) +(*pointer-‘0’)</td>
</tr>
<tr>
<td>0013</td>
<td>6 005</td>
<td>(load *05)</td>
</tr>
<tr>
<td>0014</td>
<td>2 006</td>
<td>(add 06)</td>
</tr>
<tr>
<td>0015</td>
<td>2 004</td>
<td>(add 04)</td>
</tr>
<tr>
<td>0016</td>
<td>1 004</td>
<td>(store 04)</td>
</tr>
<tr>
<td>0017</td>
<td>8 001</td>
<td>(load “01”) pointer = pointer + 1</td>
</tr>
<tr>
<td>0018</td>
<td>2 005</td>
<td>(add 05)</td>
</tr>
<tr>
<td>0019</td>
<td>1 005</td>
<td>(store 05)</td>
</tr>
<tr>
<td>001a</td>
<td>4 00a</td>
<td>(goto 0a)</td>
</tr>
<tr>
<td>001b</td>
<td>(end) 3 001</td>
<td>(halt)</td>
</tr>
</tbody>
</table>
Convert Internal Value to Numeric String

- Write ASCII character codes for value in location 5 into words 0..4 with L.S.D. in word 0.

  pointer = 0
  loop
      if pointer = 05 then quit
      if value = 0 then *pointer = ‘0’
      else *pointer = (value modulo 10) + ‘0’
      value = value / 10
      pointer = pointer + 1
  goto loop

- Exercise: write corresponding machine program; assume two new instructions
  - b xxx divide value in accumulator by value in location xxx
    and leave quotient in accumulator
  - c xxx divide by value in xxx & put remainder in accumulator
A Closer Look at Basic Computer

- **Controller** coordinates actions of other components.
- **Program Counter (PC)** stores address of next instruction.
- **Instruction Register & Decoder (IR&D)** stores current instruction.
- **Indirect Address Register (IAR)** stores indirect addresses.
- **Accumulator (ACC)** stores arithmetic operands and results.
- **Arithmetic & Logic Unit (ALU)** implements arithmetic functions.
- **Data & Address Buses** carry data between various components.
Execution of a Computer Program

reset (initialization)

system clock

time axis

monitored signals

program counter, instruction register, accumulator, ...

waveforms & buses
Execution of a Computer Program

- reset period
- fetch first instruction
- execute first instruction
- fetch second instruction
- execute second instruction