Computers and Microprocessors

Lecture 35

PHYS3360/AEP3630
Contents

• Input/output standards
• Microprocessor evolution
• Computer languages & operating systems
• Information encryption/decryption
Input/Output Ports

• **USB (universal serial bus)**
  – intelligent high-speed connection to devices
  – up to 480 Mbit/s (USB 2.0 Hi-Speed)
  – USB hub connects multiple devices
  – *enumeration*: computer queries devices
  – supports *hot swapping, hot plugging*

• **Parallel**
  – short cable, Enhanced PP up to 2 Mbit/s
  – common for printers, simpler devices
  – bidirectional, parallel data transfer (IEEE 1284)
  – Intel 8255 controller chip
Input/Output Ports (2)

• **Serial**
  – one bit at a time
  – RS-232 (recommended standard 232) serial port
  – used with long cables (not longer than ~15 m OK)
  – low speeds (up to 115 kbit/s)
  – still widely used to interface instruments
  – additional standards available:
    • E.g. RS-422/485 differential signals for better noise immunity, can support speeds in access of 10 Mbit/s (becomes cable-length dependent)
Input/Output Ports (3)

- **IEEE-488 (GPIB)**
  - Has been around for 30 years, many instruments are equipped with it
  - Allows **daisy-chaining** up to 15 devices
  - Updated versions have speeds up to 10Mbit/s
Microprocessor Evolution

- Generally characterized by the “word” size (registers and data bus)
  - 8-bit, 16-bit, 32-bit, 64-bit
  - addressable memory related to the word size
- **Intel 8080 (1974)**
  - 8-bit, *first truly usable* μ–processor (40 DIP)
  - seven 8-bit registers (six of which can be combined as three 16-bit registers)
  - 6K transistors, 2MHz clock
  - Other notable 8-bit processors include Zilog Z80 (1976) (used in Osborne 1, first portable μ-computer) and Motorola 6800/6809 (1978)
  - Small cost, compact packaging allowed **home computer revolution**
Early Intel Processors

• Intel 8080 later variants
  – 64K addressable RAM (16-bit bus address)
  – 8-bit registers
  – CP/M (control program for µ–computers) OS
  – 5,6,8,10 MHz
  – 29K transistors

• Intel 8086/8088 (1978)
  – 16-bit processor, IBM-PC used 8088
  – 1 MB addressable RAM (20-bit addresses)
  – 16-bit registers
  – 16-bit data bus (8-bit for 8088)
  – separate floating-point unit (8087)
  – used in low-cost microcontrollers now
Other 16-bit processors

- **Western Design Center (WDC 65816)**
  - used in Apple II and Super Nintendo
  - fully CMOS, low power consumption (300 $\mu$A at 1MHz, operating voltage as low as 1.8V)
  - Wait-for-Interrupt and Stop-the-Clock instructions further reduce power consumption
  - one of the most popular (made in huge numbers)
  - Still sold today (original 1984), used as a controller
  - 24-bit address bus (16MB of memory space)

- **Texas Instrument TM9900, National Semiconductor IMP-16, etc.**
IBM-AT

- **Intel 80286** (1982)
  - Still largely a *16-bit* processor
  - 16 MB addressable RAM
  - Protected memory
  - several times faster than 8086
  - introduced IDE bus architecture
  - 80287 floating point unit
  - Up to 20MHz
  - 134K transistors
32-bit processors

• **Motorola 680x0 series**
  – 32-bit registers
  – 68010 (1982) adds virtual memory support
  – Other successors **68020/68030/68040/68060**
  – Popular with UNIX operating systems in late 1980’s/early 1990’s
  – Faded from computer desktop market, but had a strong standing in embedded / controller equipment (still used)

• **“Microprocessor wars”**
  – leads to elimination of some / survival of the fittest. In aftermath, the PC market to be largely dominated by IA-32; however, much more diversity exists for the controllers
• **Intel386** (1985)
  – 4 GB addressable RAM
  – 32-bit registers
  – paging (virtual memory)
  – Up to 33MHz

• **Intel486** (1989)
  – instruction pipelining
  – Integrated FPU
  – 8K cache

• **Pentium** (1993)
  – Superscalar (two parallel pipelines)
  – Intel declines to license Pentium to others, AMD and Cyrix start their own designs
Intel Pentium Family

- **Pentium Pro** (1995)
  - advanced optimization techniques in μ-code
  - More pipeline stages
  - On-board L2 cache

- **Pentium II** (1997)
  - MMX (multimedia) instruction set
  - Up to 450MHz

- **Pentium III** (1999)
  - SIMD (streaming extensions) instructions (SSE)
  - Up to 1+GHz

- **Pentium 4** (2000)
  - NetBurst micro-architecture, tuned for multimedia
  - 3.8+GHz

- **Pentium D** (Dual core)

- …
Some interesting statistics

- 2003 data (from Wikipedia)
  - $44 billion dealt in business on microprocessors
  - Personal computers account for 50% of cost but only 0.2% of all CPU’s sold
  - 55% of all CPU’s sold are 8-bit controllers (many billions sold overall)
  - Less than 10% of all CPU’s are 32-bit or more
  - Of all 32-bit processors sold, only 2% are used in personal computers (laptops/desktops)
  - “Taken as a whole, the average price for microprocessor, microcontroller, or DSP is just over $6”
  - Read more at http://www.embedded.com/shared/printableArticle.jhtml?articleID=9900861
Programming

- Interpreted High Level Language
- Compiled High Level Language
- Assembly Language
- Machine Language
Programming (2)

• Machine Language
  – binary instructions (op codes) actually read and interpreted by the CPU
    Ex: 1000 1011 0000 0101
    ‘move value from memory to AX register’ on 386
  – different for each CPU type

• Assembly Language
  – CPU instructions represented by mnemonics
    Ex: MOV AX, M same as above
  – each AL instruction converts to one ML instruction by assembler program
  – Efficient fast execution, inconvenient to program in
  – Allows access to instructions not available with higher level language
Programming (3)

• High Level Language, e.g. C/C++
  – easier to use
  – it’s compiler’s job to translates (one) HLL instruction into (many) ML instructions
  – portable: can compile the same source (HLL instructions) on different OS platforms
  – slower and more restricted

• Interpreted Languages, e.g. JAVA, scripting
  – on-the-fly translation of high level language
  – slowest of the above, but often a good place to start with a new project
μ-processor: registers

Storage locations inside the CPU, optimized for speed.

<table>
<thead>
<tr>
<th>32-bit General-Purpose Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAX</td>
</tr>
<tr>
<td>EBX</td>
</tr>
<tr>
<td>ECX</td>
</tr>
<tr>
<td>EDX</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Control registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>EFLAGS</td>
</tr>
<tr>
<td>EIP</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>16-bit Segment Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS</td>
</tr>
<tr>
<td>SS</td>
</tr>
<tr>
<td>ES</td>
</tr>
<tr>
<td>FS</td>
</tr>
<tr>
<td>DS</td>
</tr>
<tr>
<td>GS</td>
</tr>
</tbody>
</table>

Storage locations inside the CPU, optimized for speed.
μ-processor: registers (2)

- **General Purpose Registers**: used to contain arithmetic and logical operands used by ALU
- **Segment Registers**: indicate segments of memory currently in use.
  - **CS**: (code segment) memory segment where instructions/program are located
  - **DS**: (data segment) …
  - **SS**: (stack segment) …
  - **ES**: (extra segment) …
- **Pointer & Index Registers**:
  - **BP**: base pointer
  - **SP**: stack pointer
  - **SI**: source index
  - **DI**: destination index

complete address = segment + pointer/index
μ-processor: registers (3)

- **Control Registers:**
  - IP/IR: instruction pointer held in instruction register (memory address of next instruction to be executed)
  - FLAGS: status and control flags; used to indicate processor status

<table>
<thead>
<tr>
<th>o</th>
<th>i</th>
<th>s</th>
<th>z</th>
<th>p</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

- **c = 1**: if operand produced carry
- **p = 1**: if operand has parity of 1
- **z = 1**: if result = 0
- **s = 1**: if result < 0
- **i = 1**: μ-processor will respond to interrupts
- **o = 1**: if result produced overflow
μ-processor instruction cycle

- **Fetch**: the next instruction (address is held in instruction pointer / register); instruction pointer is incremented to the next value or branched, conditional statements may throw it elsewhere

- **Decode**: what do 1’s and 0’s mean?

- **Execute**: the instruction
some basic assembler instructions

• **Data Transfer:**
  MOV d,s   move (s)ource to (d)estination

• **Arithmetic:**
  ADD d,s   add s to d and store it in d
  INC d     increment contents of d by 1

• **Logical and Shifts:**
  AND d,s   bitwise AND of s with d, store in d
  SHL d     shift d left one bit

• **Control Transfer:**
  JMP loc   jump to memory location loc
  JE loc    jump to loc if result of last operation = 0

• **I/O:**
  OUT d,s   output to the I/O space (address d)
Addressing modes

Location of the operand of an instruction may be obtained:

- **Immediate**: e.g. 200 (hex!)
  
  Operand contained in 2\textsuperscript{nd} part of the instruction

- **Register**: e.g. BX
  
  Operand is contained in one of the general registers

- **Indirect**: e.g. [200]
  
  Operand’s address is contained in the 2\textsuperscript{nd} part of the instr.

- **Register indirect**: e.g. [BX]
  
  Operand’s address is contained in one of the general pointers or pointer/index registers

- **Indexed**: e.g. [BX+1]
  
  Operand’s address is formed by adding displacement contained in 2\textsuperscript{nd} part of the instr. to the contents of one of the index registers
## Assembler example

<table>
<thead>
<tr>
<th>Addr.</th>
<th>ML</th>
<th>AL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>BB 00 02</td>
<td>MOV BX,200</td>
<td>BX ← 200h (load addr.)</td>
</tr>
<tr>
<td>103</td>
<td>8A 2F</td>
<td>MOV CH,[BX]</td>
<td>CH ← value at loc 200h</td>
</tr>
<tr>
<td>105</td>
<td>8A 4F 01</td>
<td>MOV CL,[BX+1]</td>
<td>CL ← value at loc 201h</td>
</tr>
<tr>
<td>108</td>
<td>88 0F</td>
<td>MOV [BX],CL</td>
<td>loc 200h ← CL</td>
</tr>
<tr>
<td>10A</td>
<td>88 6F 01</td>
<td>MOV [BX+1],CH</td>
<td>loc 201h ← CH</td>
</tr>
<tr>
<td>10D</td>
<td>CD 20</td>
<td>INT 20</td>
<td>software interrupt (exit)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Location</th>
<th>Value before</th>
<th>Value after</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>A1</td>
<td>B2</td>
</tr>
<tr>
<td>201</td>
<td>B2</td>
<td>A1</td>
</tr>
<tr>
<td>202</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

The program swaps values in locations 200 and 201
I/O devices usually have their own registers which are assigned or mapped to addresses in memory.

I/O is achieved by μ-processor reading from / writing to the corresponding memory addresses.

Ex: I/O user port used in the lab this week has 8 registers at addresses 2A0H → 2A7H. Used to control ADC, DAC and digital I/O functions of the port.
Operating System

• Provides a level of abstraction and security for higher level applications; interrupts, memory handling, etc.
• I/O are privileged operations, usually only OS can do I/O
• A device driver is provided, which runs as part of the OS
• User’s program then communicates to the device through the driver and OS
OS: multitasking

- Most OS can run multiple programs at the same time.
- Multiple **threads** of execution within the same program.
- **Scheduler** utility assigns a given amount of CPU time to each running program.
- Rapid switching of tasks gives illusion that all programs are running at once
- The processor must support task switching
- Scheduling policy, priority, etc.
Interrupts

Used for handling peripheral I/O asynchronously (orders of magnitude differences in time required for access, enable/disable, read/write, etc.)

Device requiring services asserts Interrupt Request.

When INTR asserted:

• µ-processor completes execution of current instruction
• IP/IR & other registers pushed onto stack
• IP/IR loaded with address of interrupt routine
• interrupt routine executed to identify and service the device
• when completed, IP & registers popped from stack, and program execution resumes
Real Time Operating Systems

General purpose OS systems such as Linux, Windows, Mac OS do not guarantee ‘real-time’ execution of instructions

i.e. they don’t necessarily have any operational deadlines from event to system response

e.g. various interrupts, multitasking, etc. are usually handled with the illusion of smooth running for a casual user, but the behavior is not deterministic

Multitasking operating systems are available that provide tools to ensure that certain deadlines from event to system response are met.

Examples: VxWorks (Wind River) – used on Mars rovers RTLinux, RTEMS (o.source)
Signal encryption using a sequence of pseudo-random numbers

- Uses linear feedback shift registers (LFSR) to generate a sequence of pseudo-random numbers (deterministic sequence that looks random)

- If LFSR has N bits, the max sequence will be $2^N - 1$ long, then repeats

- E.g. 8-bit LFSR can produce 255 long sequence, 32-bit LFSR can produce 4,294,967,295 pseudo-random sequence

- XOR gates are used to tap certain outputs into the serial input (see the table in the Supplement)
Pseudo-random number generator

- 8-bit pseudo-random generator

- Output random number is \((Q_7Q_6Q_5Q_4Q_3Q_2Q_1Q_0)_2\)
- At least one \(Q_i\) bit must be non-zero initially for non-trivial (other than always 0’s) pattern
- The initial state of \(Q_i\) bits is known as seed, which uniquely defines the sequence of pseudo-random numbers
Encrypting/decrypting

- **To encrypt** scramble the stream of data bits with pseudo-random sequence:

  \[
  EN_i = A_i \oplus PR_i
  \]

- **How to decrypt?**

  \[
  A_i = EN_i \oplus PR_i
  \]

- Need the *same unique sequence* of pseudo-random numbers used for encryption (*seed* becomes encryption/decryption password).
LTspice experiment 11.1

• Implement pseudo-random number generator
• Decrypt an encrypted 8-bit data stream
• Perform digital-to-analog conversion and plot a parametric curve \((x(t), y(t))\) to display the secret message

Homework next week

• Explain how RSA encryption/decryption works
• RSA = Rivest, Shamir, Adleman
• Public key cryptography
Arduino Uno specs

- Microcontroller: ATmega328
- Operating Voltage: 5V
- Input Voltage (recommended): 7-12V
- Input Voltage (limits): 6-20V
- Digital I/O Pins: 14 (of which 6 provide PWM output)
- Analog Input Pins: 6
- DC Current per I/O Pin: 40 mA
- DC Current for 3.3V Pin: 50 mA
- Flash Memory: 32 KB (ATmega328) of which 0.5 KB used by bootloader
- SRAM: 2 KB (ATmega328)
- EEPROM: 1 KB (ATmega328)
- Clock Frequency: 16 MHz
Different Arduinos

- Arduino Mega
- Arduino UNO
- Arduino Mini/Nano
- Lilypad