CSC 3210
Computer Organization and Programming

Introduction and Overview
Dr. Anu Bourgeois
(modified by Yuan Long)

---

Administrative Issues

• Required Prerequisites
  – CSc 2010 Intro to CSc
  – CSc 2310 Java Programming
  – CSc 2510 Discrete Math

• 2 absences allowed – otherwise could be dropped
• Read the syllabus and policies

---

Administrative Issues

• Required Textbook

- Java Architecture, Research Language Programming, and C
- The Java Virtual Machine Specification

A HTML version available at https://docs.oracle.com/javase/specs/jvms/se8/html/
Assignments

• About 5 programming assignments
• The lowest one will be dropped
• Penalty for late submissions is on the syllabus
• Must be your own work

Class Policies

• No cell phones or laptops out during class
• You may be deducted points without warning

Grading policies and Exams

• Homework (approx. 5): 20%
• Two tests: 15% each
• Final exams: 20%
• Weekly programming challenge assignments: 20%
• Attendance: 10%

All re-grading requests must be made within 2 classes from returned work

Two TAs

Lu Yao
Office: 25 Park Place (SunTrust building, 6th floor), cubicle 0650L/M/N/P/R
Email: lyao2@student.gsu.edu
Office Hour: MW 9:00 am to 10:00 am

Rohini Boosam
Office: 25 Park Place (SunTrust building, 6th floor), cubicle 7
Email: rboosam1@student.gsu.edu
Office Hour: Tuesday 1:30 pm to 2:30 pm; Thursday 1:00 pm to 2:00 pm
Expectations

- Writing code with loops
- Base conversions
  - Especially involving decimal…binary…hexadecimal
- Binary arithmetic
- Basic logic operations
- Documenting code

Why learn Assembly Language?

- Knowledge of assembly language is essential to understanding how computers are designed
- Provides the ability to optimize the code
- First word – speed
  - Gaming
  - Simulations
  - Medical equipment
- Second word – security
  - Knowing how to hack code

CSC 3210
Computer Organization and Programming

Material and images are from:
Richard P. Paul, SPARC Architecture,
Assembly Language Programming
Layout of Richard P. Paul's Chapter 1

• Hand-programmable calculator
• Fundamental definition of a computer
• Basic computer cycle
• Classic implementations of the computer
  – Stack machine architecture
  – Accumulator machine architecture
  – Load/store machine architecture

Programmable Calculators

• Numeric keyboard and function keys
• Single register – accumulator
• Arithmetic logic unit – for computations
• Stack provides memory
  – LIFO data structure
  – Pushing/popping operations
  – No addresses for the memory cells

HP-15C Programmable Calculator

Emulator available at www.hp15c.com
Postfix vs. Infix

**Postfix notation**
- Operators follow operands
  \[ 3 \ 4 \ + \]
- Uses the stack to save memory
- No need for parenthesis

**Infix notation**
- Operators are between operands
  \[ 3 \div 4 \]
- Need to specify order of operations -- parenthesis

\[
(E + \frac{F}{G})
\]

\[
A \ B + C \ D - * E \ F \ G \div \div
\]

\[
(y = (x-1) (x-7) \quad (x-11))
\]

\[
\begin{align*}
(10 - 1) &= 9 \\
(10 - 7) &= 3 \\
(9 \times 3) &= 27 \\
(10 - 11) &= -1 \\
27/(-1) &= -27 \\
\end{align*}
\]

10 enter
1 –
10 enter
7 –
*
10 enter
11 –
/
Use of Registers

Why would we want to use registers?

- Registers are provided to hold constants
- 10 registers – named r0 thru r9
- 3.14159 sto 0 – stores value in r0 and leaves it on top of stack
- rcl 0 -- copy contents of r0 to top of stack
- Must specify register name

\[ y = \frac{(x - 1)(x - 7)}{(x - 11)} \]

3.14159 sto 0  
Place the constant on the stack and store value in register r0

1 –  
Push 1, subtract, now TOP=2.14159

rcl 0  
Place value of r0 on stack, TOP=3.14159

7 –  
Push 7, subtract, TOP= -3.8584

*  
Multiply, TOP = -8.2631

rcl 0  
Place value of r0 on stack, TOP=3.14159

11 –  
Push 11, subtract, TOP = -7.8584

/  
Divide, TOP = -0.6515
Programmable Calculators

- In program mode, keystrokes not executed, code for each key is stored in memory
- Memory has an address and holds data
- Principal key designation
- Function keys
- Machine language – codes for keystrokes
- Central processing unit
- Program counter – holds address of next instruction to be executed

\[ y = \frac{(x - 1)(x - 7)}{(x - 11)} \]

*sto 0* Store the number(s) entered in register 0, leaving a copy of it on the top of the stack
*1 –* Push 1, subtract, TOP=x-1
*rol 0* Place value of r0 on top of stack, TOP=x
*7 –* Push 7, subtract, TOP=x-7
*\times* Multiply, TOP=(x-1)*(x-7)
*rol 0* Place value of r0 on stack, TOP=x
*11 –* Push 11, subtract, TOP=x-11
/* Divide, TOP = (x-1)*(x-7)/(x-11)

Have a try?

\[ y = \frac{(x - 1)(x - 7)}{(x - 11)} \]

\text{g P/R}
\text{f LBL A}
\text{.....(Put the program here)}
\text{g RTN}
\text{g R/S}
\text{3.172843}
\text{GSB A}

? How to write a program on HP15C

Please refer to

http://hp-15c.homepage.t-online.de/
programs.html#prgmmod
Memory

- Memory used to store program
- Memory is addressed
- May compute memory addresses – unlike registers
- Registers may be selected – not indexed

Machine language

- Program stored using machine language – key codes of the calculator
- Central processing unit (CPU) executes the codes
- Program counter (PC) holds address of next instruction to be executed
<table>
<thead>
<tr>
<th>Address</th>
<th>M/C code</th>
<th>Keystrokes</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>000 - 001</td>
<td>44 0</td>
<td>sto 0</td>
<td>Store in register 0</td>
</tr>
<tr>
<td>002</td>
<td>1</td>
<td>1</td>
<td>Enter 1</td>
</tr>
<tr>
<td>003</td>
<td>30</td>
<td>–</td>
<td>Subtract</td>
</tr>
<tr>
<td>004 - 005</td>
<td>45 0</td>
<td>rel 0</td>
<td>Register 0 to stack</td>
</tr>
<tr>
<td>006</td>
<td>7</td>
<td>?</td>
<td>Enter 7</td>
</tr>
<tr>
<td>007</td>
<td>30</td>
<td>–</td>
<td>Subtract</td>
</tr>
<tr>
<td>008</td>
<td>20</td>
<td>*</td>
<td>Multiply</td>
</tr>
<tr>
<td>009 - 010</td>
<td>45 0</td>
<td>rel 0</td>
<td>Register 0 to stack</td>
</tr>
<tr>
<td>011</td>
<td>1</td>
<td>1</td>
<td>Enter 1</td>
</tr>
<tr>
<td>012</td>
<td>1</td>
<td>1</td>
<td>Make it 11</td>
</tr>
<tr>
<td>013</td>
<td>30</td>
<td>–</td>
<td>Subtract</td>
</tr>
<tr>
<td>014</td>
<td>10</td>
<td>/</td>
<td>Divide</td>
</tr>
<tr>
<td>015 - 016</td>
<td>43 32</td>
<td>g Rtn</td>
<td>Return to calculator mode</td>
</tr>
</tbody>
</table>

- Calculator mode – codes (m/c lang.) sent to ALU
- Program mode – codes (m/c lang.) sent to memory
  - Each machine code is stored in one addressable memory location

Von Neumann Machine
- Contains addressable memory for instructions and data
- ALU executes instructions fetched from memory
- PC register holds address for next instruction to execute
- Defined an instruction cycle
Instruction Cycle

```c
pc = 0;
do {
    instruction = memory[pc++];
    decode (instruction);
    fetch (operands);
    execute;
    store (results);
} while (instruction != halt);
```

Memory

---

Figure 5.3 Structure of Random Access Memory
The Components of a Computer System

- Von Neumann architecture is based on the following three characteristics
  - Four major subsystems called memory, input/output, the arithmetic/logic unit (ALU), and the control unit
  - The stored program concept
  - The sequential execution of instructions
Machine Language Instructions

- Instructions that can be decoded and executed by the control unit of a computer
- Operation code (Operator) field
  - Unique unsigned integer code assigned to each machine language operation recognized by the hardware
- Operand field(s)
  - A number
  - Memory addresses of values on which the operation will work
  - Variable name
    Depends on a specified operator and assembler
Machine Language Instructions (continued)

- Instruction set
  - Set of all operations that can be executed by a processor
- Reduced instruction set computers or RISC machines
  - Include as little as 30–50 instructions
- Complex instruction set computers (CISC machines)
  - Include 300–500 very powerful instructions

Machine Language Instructions (continued)

- Classes of machine language instructions
  - Data transfer
  - Arithmetic
  - Compare
  - Branch

Control Unit Registers and Circuits

- Program counter (PC)
  - Holds the address of the next instruction to be executed
- Instruction register (IR)
  - Holds a copy of the instruction fetched from memory
- Instruction decoder
  - Determines what instruction is in the IR
Putting All the Pieces Together—the Von Neumann Architecture

- Program execution phases
  - Fetch, decode, and execute
- Von Neumann cycle
  - The repetition of the fetch/decode/execute phase
Stack Machine

- Stack architecture does not have registers
- Use memory to place items onto stack
- Use push and pop operations for moving data between memory and the stack
- Must specify memory address
- MAR – memory address register
- MDR – memory data register
- IR – instruction register holds fetched instruction
- ALU uses top two elements on the stack for all computations

Assume address 300 holds the value 3 and address 400 holds the value 4
push [300]
push [400]
add
pop [500]
Accumulator Machine

- Accumulator register used as source operand and destination operand
- Use load and store operations to move data from accumulator from/to memory
- No registers or stack
- Must access memory often

Load Store Machine

- Initially memory limited to few hundred words
- Access time to all locations was the same
- As memory size increased time vs. cost issue arose
- New designs included variable access times
- Register file – high speed memory
Load Store Machine

- Use *load* and *store* instructions between registers and memory
- ALU functions on registers only
- Register file replaces the stack of the stack machine
- SPARC architecture is a load/store machine

Assemblers

- An assembler is a macro processor to translate symbolic programs into machine language programs
- Symbols may be used before they are defined – unlike using m4
- Two pass process
  - Once to determine all symbol definitions  **First Pass**
  - Once to apply the definitions  **Second Pass**
Symbols

- A symbol followed by a colon defines the symbol to have as its value the current value of the location counter
- The symbol is called a label

First Pass

![Image of First Pass](image1.png)

Second Pass

![Image of Second Pass](image2.png)
define(y_r, r0)
define(x_r, r1)
define(a2_r, r2)
define(a1_r, r3)
define(a0_r, r4)
define(temp_r, r5)

\[
\begin{align*}
\text{start: } & \quad \text{mov 0, \%x_r} \quad ! \text{initialize x = 0} \\
& \quad \text{mov a2, \%a2_r} \\
& \quad \text{mov a1, \%a1_r} \\
& \quad \text{mov a0, \%a0_r} \\
& \quad \text{sub \%x_r, \%a2_r, \%y_r} \quad ! (x-1) \\
& \quad \text{sub \%x_r, \%a1_r, \%temp_r} \quad ! (x-7) \\
& \quad \text{mul \%y_r, \%temp_r, \%y_r} \quad ! (x-1) \cdot (x-7) \\
& \quad \text{sub \%x_r, \%a0_r, \%temp_r} \quad ! (x-11) \\
& \quad \text{div \%y_r, \%temp_r, \%y_r} \quad ! \text{divide to compute y}
\end{align*}
\]