CSC 3210
Computer Organization and Programming

Introduction and Overview
Dr. Anu Bourgeois
(modified by Michael Weeks)

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

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

• Required Textbook
  TBA

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

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

Assignments
• About 5 programming assignments
• 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

CSC 3210
Computer Organization and Programming

Material and images are from:
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

Postfix vs. Infix

**Postfix notation**
- Operators follow operands
- Uses the stack to save memory
- No need for parenthesis

**Infix notation**
- Operators are between operands
- Need to specify order of operations -- parenthesis

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

![HP-15C Programmable Calculator](image)

Emulator available at www.hp15c.com

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

\[ \frac{(A + E) \cdot (C - D)}{E + \frac{F}{G}} \]

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

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

10 enter
1
10 enter
7
- 
*
10 enter
11
- 
/
Stack Operations

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

Memory

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

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

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

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

3.14159 sto 0  Place the constant on the stack and store it in register r0
1 – Push 1, subtract, now TOP=2.14159
rel 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 = 1.0515
### 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
```plaintext
pc = 0;
do {
  instruction = memory[pc++];
  decode (instruction);
  fetch (operands);
  execute;
  store (results);
} while (instruction != halt);
```

### 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
Stack Machine

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

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

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

• 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

Accumulator Machine

Assume address 300 holds the value 3 and address 400 holds the value 4
load 300
add 400
store 500

Load Store Machine

Assume address 300 holds the value 3 and address 400 holds the value 4
load [300], r0
load [400], r1
add r0, r1, r0
store r0, [500]
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
  - Once to apply the definitions

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

define(y_r, r0)
define(x_r, r1)
define(a2_r, r2)
define(a1_r, r3)
define(a0_r, r4)
define(temp_r, r5)

start:  mov 0, %x_r         ! initialize x = 0
        mov a2, %a2_r
        mov a1, %a1_r
        mov a0, %a0_r
        sub %x_r, %a2_r, %y_r  ! (x-1)
        sub %x_r, %a1_r, %temp_r  ! (x-7)
        mul %y_r, %temp_r, %y_r  ! (x-1)*(x-7)
        sub %x_r, %a0_r, %temp_r  ! (x-11)
        div %y_r, %temp_r, %y_r  ! divide to compute y