CSC 3210
COMPUTER ORGANIZATION AND PROGRAMMING

POLICY AND SYLLABUS OVERVIEW
CHINUA UMJOA
INSTRUCTOR AND TEACHING ASSISTANTS

• **Chinua Umoja – Instructor**
  • cumoja1@gsu.edu
  • Desk Number – 0641C
  • Lab Number – TBD
  • Office Hours – MW / 2:00 – 3:00pm and by appointment

• **Saumya Jain - TA**
  • sjain4@student.gsu.edu
  • Desk Number – TBD
  • Office Hours – TR / Noon – 2:00pm and by appointment

• **Hima Govada - TA**
  • hgovada1@student.gsu.edu
  • Desk Number – TBD
  • Office Hours – MW/ 4:00 – 5:30pm and by appointment

SUBJECT TO CHANGE
PREREQUISITES

CSC 2310 (Principles of Computer Programming I)
CSC 2510 (Theoretical Foundations of Computer Science)

A basic knowledge of pseudo code, common coding practice, binary number systems and there operations (logical and arithmetic)
COURSE DESCRIPTION

We discuss the computer and machine structure, machine language, SPARC assembly language, addressing techniques, macros, program segmentation and linkage.

TEXT:

"SPARC Architecture, Assembly Language Programming, and C,"
PLANNED CHAPTERS

• **Chapter 1 – Introduction to Computer Architecture**
  • Machine code, post/in/pre order notation, assembly architecture

• **Chapter 2 – SPARC Architecture**
  • Introduction into SPARC, variables, loops, control statements, and the GDB debugger

• **Chapter 3 – Digital Logic and Binary Numbers**
  • Number Conversion, logical operators

• **Chapter 4 – Binary Arithmetic**
  • Complements, subtraction and addition of negative numbers, multiplication and division

• **Chapter 5 – The Stack**
  • Addressing the memory

• **Chapter 7 – Subroutines**
  • Calling methods

• **Chapter 8 – Machine Instructions**
  • Converting code to its binary and hexadecimal equivalent
ATTENDANCE

Student is allowed up to two (2) absences. Roll will be taken during class, and a late student will be counted as absent. If a student has greater than three absences, then the instructor may drop the student from the class or drop the final grade by one letter grade. Any student missing a lesson is responsible for all material assigned or covered in class during his or her absence. Do not send me an e-mail asking what was covered in the class missed.
GRADING

The grades for this course will be based upon the following components and will be awarded based upon the 10 point scale:

1. Exams (approx. 3) 40%
2. Final Exam (December 15th 1:30 – 4:00) 20%
3. Quizzes (approx. 5) 10%
4. Programming Assignments 25%
5. Chapter Assignments 5%
HOMEWORK ASSIGNMENTS

Grades By TAs
Due 10 days after assignment
Turned in on D2L (and class server if coding involved)
PROGRAMMING ASSIGNMENTS

Will be graded by TAs.
Will be Assigned on Wednesdays and due 16 days after they have been assigned.
Turned in on D2L and class server
STUDENTS WITH DISABILITIES

The Special Services Disabilities Program Office provides counseling and advocacy for eligible students with disabilities (auditory, visual, orthopedic, medical, or learning disabilities). Requests for special accommodations need to be submitted to the lecturer at the beginning of the course in order to adequately meet student needs. The Special Services Program Office (Room 230 Student Center. Phone 404 413-1560) has the proper forms you need to turn in to instructors.
1. Make-up exams must be arranged prior to the scheduled date and will be allowed only at the discretion of the instructor. There will be no make-up quizzes.

2. All assignments must be turned by the start of class. Assignments turned in later in class, even if on the date of the deadline, will be considered late.

3. The penalty for late assignment submissions is 20% and must be turned in by the start of the next class following the deadline. No assignments will be accepted after this time.

4. Any material submitted for work should be the student’s own work. Duplicate assignments will receive a grade of zero. Any student found to be cheating on any graded work will receive a score of zero. The Dean of Students office will also be notified. Refer to the university policy.
5. All re-grading requests must be made within two classes of when the work was first returned in class.

6. Cell phones must be turned to silent during class – this includes no texting. First violation receives a warning. All succeeding violations result in a five point deduction off the final exam. Any violation during a quiz or exam results in a ten percent deduction off the corresponding paper. No warnings for quizzes or exams.

7. All exams and quizzes will be proctored by an experienced instructor.

8. Seat assignments may be issued for any exams or quizzes, as may multiple, but equivalent papers.

9. Different room assignments may be used for exams.
LAYOUT OF 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
  - $3 + 4$
- Need to specify order of operations – parenthesis
\[
\frac{(A + B) * (C - D)}{E + \frac{F}{G}}
\]

\[A B + C D - * E F G \div + \div\]
\[ y = (x-1)(x-7)(x-11) \]

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

10 enter
1 −
10 enter
7 −
*  
10 enter
11 −
/
STACK OPERATIONS

Figure 1.2: The Evaluation of $y = (10 - 1)(10 - 7)/(10 - 11)$
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
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
\[ y = \frac{(x - 1)(x - 7)}{(x - 11)} \]

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 = 1.0515
MEMORY

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

struct registers {
    int r0, r1, r2, r3, r4, r5, r6, r7, r8, r9;
}
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>rcl 0</td>
<td>Register 0 to stack</td>
</tr>
<tr>
<td>006</td>
<td>7</td>
<td>7</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>rcl 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
MACROS

• Macro processor m4 – translates symbols into numeric constants

Macros defined using define macro – two arguments

define(sto, 44 0)
define(rcl, 45 0)
define(div, 10)
Macros may have up to 9 arguments
Specify arguments by $n$
If macro name followed immediately by ‘(‘, then arguments are present

define(cat, $1$2$3$4$5)
call it by: cat(a, b, c, d, e) $\Rightarrow$ ab cde
call it by: cat(a, b, c) $\Rightarrow$ a b c
define(sto, 44 0) then sto always refers to r0

define(sto, ’44 $1’) then call it as sto(0)

-This makes code easier to read
-Macros are essentially substitutions that can use arguments
define(f, 42)
define(g, 43)
define(loc, 0)
define(sto, 'loc: 44 $1 define( 'loc', eval(loc + 2))')
define(rcl, 'loc: 45 $1 define( 'loc', eval(loc + 2))')
define(div, 'loc: 10 define( 'loc', eval(loc + 1))')
define(mul, 'loc: 20 define( 'loc', eval(loc + 1))')
define(label, 'define($1, loc)')
define(ifeq, 'loc g 20 define( 'loc', eval(loc + 2))')
define(gto, 'loc 22 $1 define( 'loc', eval(loc + 2))')
<table>
<thead>
<tr>
<th>Address</th>
<th>M/C code</th>
<th>Assembly Code</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>44 0</td>
<td><code>sto(0)</code></td>
<td>Store in register 0</td>
</tr>
<tr>
<td>002</td>
<td>1</td>
<td><code>digit(1)</code></td>
<td>Enter 1</td>
</tr>
<tr>
<td>003</td>
<td>30</td>
<td><code>Sub</code></td>
<td>Subtract</td>
</tr>
<tr>
<td>004</td>
<td>45 0</td>
<td><code>rcl(0)</code></td>
<td>Register 0 to stack</td>
</tr>
<tr>
<td>006</td>
<td>7</td>
<td><code>digit(7)</code></td>
<td>Enter 7</td>
</tr>
<tr>
<td>007</td>
<td>30</td>
<td><code>Sub</code></td>
<td>Subtract</td>
</tr>
<tr>
<td>008</td>
<td>20</td>
<td><code>Mul</code></td>
<td>Multiply</td>
</tr>
<tr>
<td>009</td>
<td>45 0</td>
<td><code>rcl(0)</code></td>
<td>Register 0 to stack</td>
</tr>
<tr>
<td>011</td>
<td>1</td>
<td><code>digit(1)</code></td>
<td>Enter 1</td>
</tr>
<tr>
<td>012</td>
<td>1</td>
<td><code>digit(1)</code></td>
<td>Make it 11</td>
</tr>
<tr>
<td>013</td>
<td>30</td>
<td><code>sub</code></td>
<td>Subtract</td>
</tr>
<tr>
<td>014</td>
<td>10</td>
<td><code>div</code></td>
<td>Divide</td>
</tr>
<tr>
<td>015</td>
<td>43 32</td>
<td><code>g rtn</code></td>
<td>Return to calculator mode</td>
</tr>
</tbody>
</table>
Write the machine code and assembly code to compute:

\[ y = \frac{3x + \frac{x}{5}}{(x - 24)(x + 9)} \]
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
VON NEUMANN MODEL

CPU

ALU

PC

Registers

Data lines
Address lines
Control lines

Memory

I/O
INSTRUCTION CYCLE

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
Assume address 300 holds the value 3 and address 400 holds the value 4

**Push** [300]

**Push** [400]

**Add**

**Pop** [300]
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
Assume address 300 holds the value 3 and address 400 holds the value 4

load [300]
add [400]
store [300]
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
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, [300]
```
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
HOMEWORK

MANDATORY
1 – 6
13 – 16

EXTRA
7 – 12