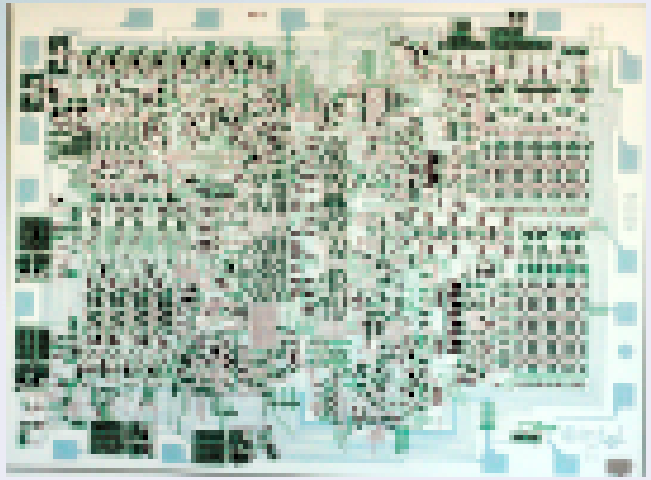

Lecture 8b: Programming the LC-3 in Assembly Language

CSIS11: **Computer Architectures and Organization**

Readings

- *Chapter 8* Patt and Patel: Introduction to Computing Systems...
- *Guide to Using LC-3 Tools* - *handout*
- *Appendix A* Patt and Patel: Introduction to Computing Systems... - *handout*
- C to LC-3 Excellent source for comparing *C* to *LC-3* syntax

Monday Class - Check in Notes

1. **AI/CoPilot can be wrong, very wrong and very annoying.**
2. Make sure you *Commit/Sync* often, it saves your work and updates your homework in GitHub
3. Be sure to update the right *README*! So its easy **for me** to see whose repo...
4. Add the LC3 Extension to VS Code to get code highlighting and easy commenting
5. DO NOT EDIT CODE IN LC3Tools, this is a bad practice, use VS Code and copy/paste

AI Issues

1. LC-3 Assembly language is a bit esoteric, which means an LLM won't have a complete grasp of it
2. Microsoft's CoPilot is too aggressive, which means it regularly recommends stuff which flat out **do not work**
3. Remember my rant about "Year of the LAN", these issues occur every time a technology appears to be a step function...*personal computing*, *LAN*, *Internet*, *AI*

So what to do?

Continue to develop critical thinking skills which help you to understand where AI might be successful (or accurate) and where it will have issues, this is where you will make money.

Programming Tips

1. Use the stack.
2. Keep track of register values
3. Use multiply to account for SIZEOF in tables
4. Use pointers.
5. Use subroutines

Use the stack

```
; stack_example - stack include code as an example
; to use stack copy and paste "stack include section" code into your program
; Be aware of the following:
;   1. The stack is a 4 entry stack, with the stack pointer starting at x4000
;   2. Change FULL to be a lower number for a larger stack
;   3. To use the stack, call STK_INIT before using the stack
;   4. To push a value onto the stack, load the value into R0 and call PUSH
;   5. PUSH will return with R5 = 0 if successful, R5 = 1 if the stack is full
;   6. To pop a value from the stack, call POP, the result will be in R0
;   7. POP will return with R5 = 0 if successful, R5 = 1 if the stack is empty
```

Example code

```
; Example code:
                .ORIG x3000
                JSR STK_INIT    ; Initialize the stack
;  ...code that places a value into R0 which you want to reuse
                JSR PUSH        ; PUSH R0 onto the stack
                BRnp ERR        ; always call BRnp ERR after PUSH to check for success
;  code that does something else with R0
                JSR POP         ; Get DATA back into R0
                BRnp ERR        ; always call BRnp ERR after PUSH to check for success
; ...resume code that uses the original R0
                HALT
ERR             LEA R0,ERRMSG   ; Load the error message
                PUTS            ; Print the error message
                HALT
ERRMSG          .STRINGZ "Stack error\n"
```

Keep track of complemented values

```
...code prior to this put x-30 and x-39 back into ASCII_0 and ASCII_9
INPUT   GETC
        LD R3, ASCII_9
        ADD R2, R0, R3  ; check if > ASCII_9
        BRp INPUT       ; if so, get another character

        LD R3, ASCII_0
        ADD R2, R0, R3 ; check if < ASCII_0
        BRn INPUT       ; if so, get another character

        ADD R0, R0, R3 ; convert to integer, by subtracting ASCII 0
```

Use multiply to account for SIZEOF (table)

```
SIZEOF .FILL 10          ; the size of each string in the array
NUMS    .STRINGZ "nullus   "
        .STRINGZ "unus     "
        .STRINGZ "duo      "
        .STRINGZ "tres     "
        .STRINGZ "quattuor "
        .STRINGZ "quinque  "
        .STRINGZ "sex      "
        .STRINGZ "septem   "
        .STRINGZ "octo     "
        .STRINGZ "novem    "
```

Use multiply to account for SIZEOF (code)

```
...R0 contains original value to multiply...
        LD R4, SIZEOF
        AND R3, R3, #0  ; clear R3

MULT    ADD R3, R3, R0  ; adder for multiplication
        ADD R4, R4, #-1 ; decrement multiplier count
        BRp MULT        ; loop if not done

        LEA R1, NUMS    ; R1 is the address of the beginning of the array
        ADD R0, R3, R1  ; R0 contains address of the I-th element
        PUTS            ; print out the Latin word for the number
```

Use pointers.

```
        LEA R1, NUMS    ; R1 has address of the beginning of the array
        ADD R0, R3, R1  ; R3 has the offset -> R0 is address of the I-th element
        PUTS            ; print out the Latin word for the number
```

JSR vs. JSRR

What is the difference between JSR and JSRR and why is this important?

```
JSR QUEUE   ; Put the address of the instruction following JSR into R7
            ; Jump to QUEUE (PCoffset11).
JSRR R3     ; Put the address following JSRR into R7
            ; Jump to the address contained in R3.
```

JSRR allows for Dynamic Jumps based on a register value

Simple algorithm

1. Input a character
2. If character is "+", subroutine PLUS
3. If character is "-", subroutine MINUS
4. If character is "x", subroutine MULT
5. If character is "/", subroutine DIVD
6. else, subroutine ERROR

Postfix Notation (*Reverse Polish Notation*)

1. Put a number on the stack - 5
2. Put a number on the stack - 4
3. Put a operand (+, -, x, /) on the stack - +
4. Put a number on the stack - 9
5. Put a operand (+, -, x, /) on the stack - x
6. Put a number on the stack - 6
7. Put a number on the stack - 3
8. Put a operand (+, -, x, /) on the stack - -
9. Put a operand (+, -, x, /) on the stack - /
10. Equivalent -> ((5 + 4) x9) / (6-3)

RPN	5	4	+	9	*	6	3	−	/
							3		
		4		9		6	6	3	
Stack	5	5	9	9	81	81	81	81	27

Begin to Develop Algorithm for SMC

1. Ask for four numbers
2. Convert to decimal value
3. Ask for four numbers
4. Convert to decimal value
5. Ask for operation
6. Perform operation
7. Output ASCII Equiv

Links

- Reverse Polish Notation