# Constructing an Assembler and Virtual Machine

Devin Wright
NCUR - 2021

## Assembler/Virtual Machine

- **Create a 2 pass Assembler**
- **Create a Virtual Machine**
- **Test virtual machine with simple program performing math operations and printing characters/numbers**

### Assembler Pass 1 Tokenizer

- Tokenize and load symbols into the symbol table.
- Simple syntax errors reported

```
1   COUNT    .INT   1
2   INDEX    .INT   0
3   NEXT     .INT   5
4   R        .BYT   'R'
5   A        .BYT   'A'
6   Y        .BYT   'Y'
7   START    LDR    R0  COUNT
8            LDR    R1  INDEX
9            ADD    R1  R0
10           STR    R1  INDEX
11           TRP    1
12  BEGIN    MOV    R3  R0
13           CMP    R3  R5
14           BNZ    R3  BADX
15           TRP    3
16           TRP    0
```

**Symbol Table**

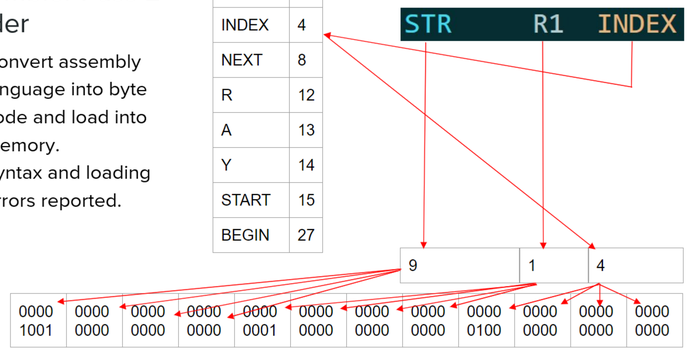
| | |
|---|---|
| COUNT | 0 |
| INDEX | 4 |
| NEXT | 8 |
| R | 12 |
| A | 13 |
| Y | 14 |
| START | 15 |
| BEGIN | 27 |

### Assembler Pass 2 Loader

- Convert assembly language into byte code and load into memory.
- Syntax and loading errors reported.

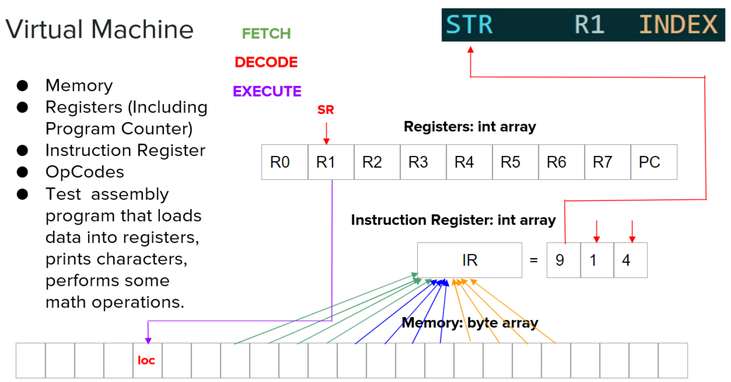
**Symbol Table**

| | |
|---|---|
| COUNT | 0 |
| INDEX | 4 |
| NEXT | 8 |
| R | 12 |
| A | 13 |
| Y | 14 |
| START | 15 |
| BEGIN | 27 |




### Virtual Machine

- Memory
- Registers (Including Program Counter)
- Instruction Register
- OpCodes
- Test assembly program that loads data into registers, prints characters, performs some math operations.

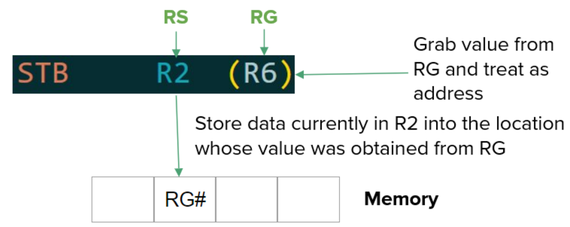



The "Big Switch"

```
while (running) {
    // Fetch
    Fetch(); // PC incremented after fetch
    // Decode
    switch: (IR[0])
    {
        case 24:
            // Execute
            ExecuteSTBIndirect();
            break;
    }
}
```

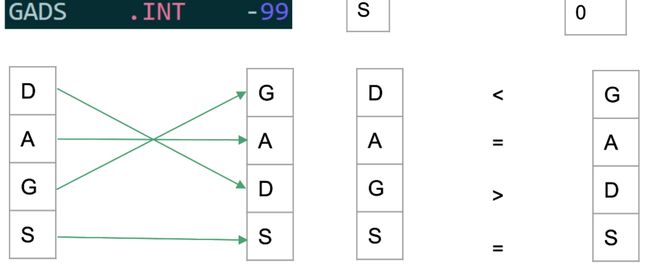
## Register Indirect Addressing, Loops, and Conditionals

- **Add more opcodes for jump, move, and compare instructions.**
- **Make VM capable of Register Indirect Addressing.**
- **Show VM capability by developing an assembly program that implements:**
  - **Array Traversal**
  - **Loops**
  - **Conditional (if, else) statements**
  - **Demonstrate byte addressability by accessing chars, switching values, and performing various comparisons.**




```
DAGS    .BYT   'D'
        .BYT   'A'
        .BYT   'G'
        .BYT   'S'
GADS    .INT   -99
```

| | |
|---|---|
| D | -99 |
| A | 0 |
| G | 0 |
| S | 0 |



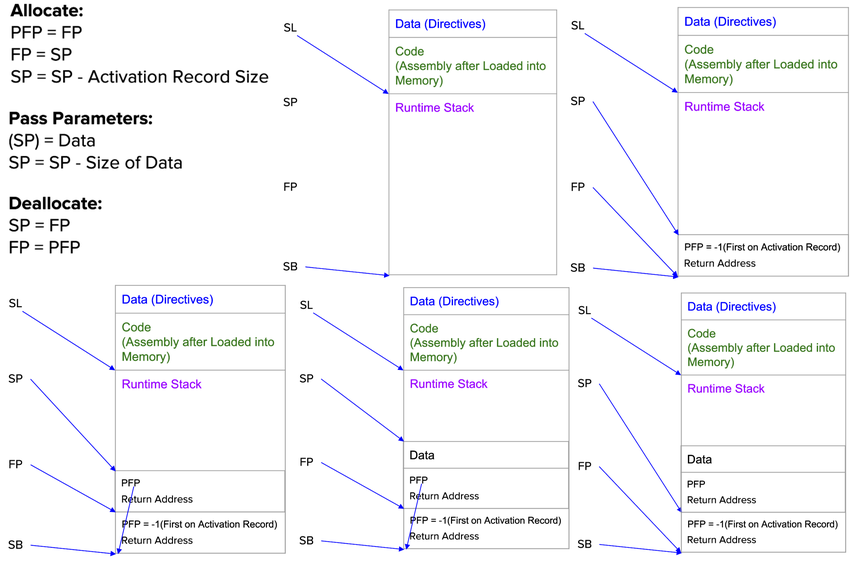

## Functions and Runtime Stack

- **Implement the Runtime Stack**
- **Implement a large assembly program managing Activation Records on the Runtime Stack for function calls..**

**Allocate:**
PFP = FP
FP = SP
SP = SP - Activation Record Size

**Pass Parameters:**
(SP) = Data
SP = SP - Size of Data

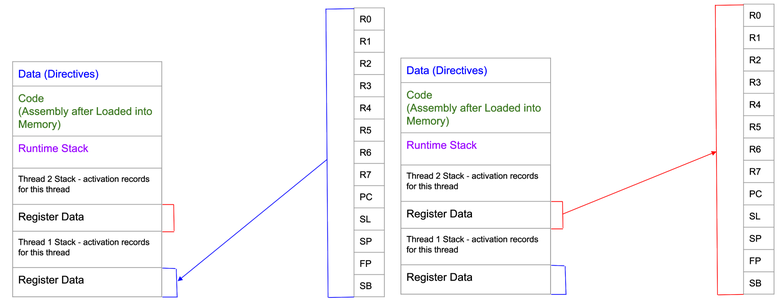
**Deallocate:**
SP = FP
FP = PFP




## Recursion and Multithreading

- **Demonstrate Virtual Machine's ability to handle recursion.**
- **Implement Multithreading.**

```
;; Test for overflow (SP < SL)
;; create activation record
JMP     label
label
;; Base Case if true JMP RETURN
JMP RETURN
;; Test for overflow (SP < SL)
;; create activation record
JMP     label
RETURN
;; Test for underflow
;; deallocate current activation record
;; return
;; Put return address into Register
;; Point at previous activation record
;; return to return address in Register
```




The "Big Switch" + Context Switch

```
while (running) {
    // Fetch
    Fetch(); // PC incremented after fetch
    // Decode
    switch: (IR[0])
    {
        case 24:
            // Execute
            ExecuteSTBIndirect();
            break;
    }

    // First copy all registers to memory
    for (int i = 0; i < NUMREGISTERS; i++)
    {
        Memory[address] = registers[i];
        tempSB++;
    }

    // round robin
    currentThread = (currentThread + 1) % 5;
    while(runningThreads[currentThread] == false)
    {
        currentThread = (currentThread + 1) % 5;
    }

    // Copy data from memory to registers
    int tempSB = SB - (THREADSIZE);

    for (int i = 0; i < NUMREGISTERS; i++)
    {
        registers[i] = Memory[tempSB];
        tempSB++;
    }
}
```

## CONCLUSIONS

- **Fully functional 2 pass and Assembler and Virtual Machine capable of Functions, Recursion, Arrays, and Multithreading.**
- **Currently developing a Compiler to work directly on my Assembler/Virtual Machine..**

## ACKNOWLEDGEMENTS

**I'd like to thank my mentor Dr. Curtis Ray Welborn for his guidance and support throughout this project.**
**Thank you to all who came to learn about my Assembler and Virtual Machine.**