Is There No End to Microcode?

COMP375
Computer Architecture and Organization
“Friendship that insists upon agreement on all matters is not worth the name. Friendship to be real must ever sustain the weight of honest differences, however sharp they be.”

Mahatma Gandhi
Schedule

• Microcode homework due **Wednesday**, October 2, 2019
# Table and Switches

<table>
<thead>
<tr>
<th>bus</th>
<th>IR adr</th>
<th>bus</th>
<th>resu lt</th>
<th>A L U</th>
<th>bus</th>
<th>PC</th>
<th>bus</th>
<th>R1</th>
<th>bus</th>
<th>R2</th>
<th>bus</th>
<th>M A R</th>
<th>bus</th>
<th>M B R</th>
<th>bus</th>
<th>ALU func</th>
<th>Mem func</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>X</em></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Diagram:**
- IR (Instruction Register) → IR addr → result
- ALU (Arithmetic Logic Unit) → result
- PC (Program Counter) → bus
- R1, R2, MBR (Memory Buffer) → bus
- MAR (Memory Address Register) → bus
- Memory System

**Table:**
- 2 ways to address the table:
  - Top row (bus): IR, IR addr, result, A L U, bus, PC, bus, R1, bus, R2, bus, M A R, bus, M B R, bus
  - Columns (bus): IR, IR adr, result, A L U, bus, PC, bus, R1, bus, R2, bus, M A R, bus, M B R, bus, ALU func, Mem func

**Notes:**
- An *X* indicates a connection or flow of data.
Microcode Store

Microcode address register

Microcode buffer register

clock
memory wait
memory ready

to control switches
Simple CPU
The READ memory function is used

A. To get data from RAM
B. To get data from a register
C. All of the above
D. None of the above
Follow the Fetch/Execute Cycle

• The steps of the fetch/execute cycle are reflected in the microcode
  1. Read the instruction from memory
  2. Increment the program counter
  3. Get the operands
  4. Execute the instruction
  5. Save the results
Fetch and PC Increment

• The fetch/execute cycle always starts with reading the instruction from memory and incrementing the program counter
• In our simple computer, this takes three lines of microcode
• After the microcode for one instruction the CPU will start the fetch of the next instruction
Jump Instructions

• The last microcode step of a jump almost always copies a value into the program counter
• Jump instructions *rarely* access memory unless they are pushing or popping something on the stack
Arithmetic

• Most arithmetic instructions have the steps
  – Copy something into the operand reg.
  – Put a value on the bus and do it.
  – Copy the result register someplace

• In a more realistic system, the ALU function would be determined by the opcode of the instruction
Register Indirect with Offset

- The address of the data is the sum of the instruction offset field and a register value
- Useful when addressing an array

Diagram:
- Instruction
- Data register
- Address
- Data
- Memory
How many times will a sub R1, cow[R2] instruction use the ALU?

A. 1
B. 2
C. 3
D. none
Register Indirect with Offset
Add to R1 using Register Indirect with Offset with R2 as the index

![Diagram of memory system and ALU]

<table>
<thead>
<tr>
<th>bus →</th>
<th>IR →</th>
<th>result →</th>
<th>bus →</th>
<th>bus →</th>
<th>PC →</th>
<th>bus →</th>
<th>R1 →</th>
<th>bus →</th>
<th>R2 →</th>
<th>bus →</th>
<th>bus →</th>
<th>M →</th>
<th>A →</th>
<th>Mem func</th>
</tr>
</thead>
<tbody>
<tr>
<td>IR</td>
<td>IR</td>
<td>result</td>
<td>IR</td>
<td>IR</td>
<td>PC</td>
<td>IR</td>
<td>R1</td>
<td>IR</td>
<td>R2</td>
<td>IR</td>
<td>IR</td>
<td>M</td>
<td>A</td>
<td>func</td>
</tr>
<tr>
<td>addr</td>
<td>addr</td>
<td>bus</td>
<td>R1</td>
<td>R1</td>
<td>bus</td>
<td>R1</td>
<td>bus</td>
<td>R2</td>
<td>bus</td>
<td>M</td>
<td>M</td>
<td>L</td>
<td>U</td>
<td></td>
</tr>
<tr>
<td>bus</td>
<td>bus</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td></td>
</tr>
<tr>
<td>oprd</td>
<td>oprd</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td></td>
</tr>
<tr>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td></td>
</tr>
<tr>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td></td>
</tr>
<tr>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td></td>
</tr>
<tr>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td></td>
</tr>
<tr>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td></td>
</tr>
<tr>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td></td>
</tr>
</tbody>
</table>

X = Address

- add
- read
- wait
- Mem func: Memory function
Register Indirect with Offset
Add to R1 using Register Indirect with Offset with R2 as the index
Register Indirect with Offset
Add to R1 using Register Indirect with Offset with R2 as the index

![Diagram of register indirect with offset]

<table>
<thead>
<tr>
<th>bus</th>
<th>IR</th>
<th>result</th>
<th>bus</th>
<th>R1</th>
<th>bus</th>
<th>bus</th>
<th>R2</th>
<th>bus</th>
<th>A</th>
<th>L</th>
<th>U</th>
<th>M</th>
<th>B</th>
<th>R</th>
<th>A</th>
<th>L</th>
<th>U</th>
<th>Mem func</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>add</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>read</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>wait</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>add</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Register Indirect with Offset
Add to R1 using Register Indirect with Offset with R2 as the index

<table>
<thead>
<tr>
<th>bus → IR</th>
<th>IR addr → bus</th>
<th>result → bus</th>
<th>bus → ALU</th>
<th>bus → PC</th>
<th>bus → R1</th>
<th>bus → R2</th>
<th>bus → MBR</th>
<th>bus → MBR</th>
<th>M BR → bus</th>
<th>A LU fun</th>
<th>Mem func</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>add</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>read</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>wait</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>add</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Diagram:

- IR: Instruction Register
- IR addr: Instruction Register Address
- ALU: Arithmetic Logic Unit
- PC: Program Counter
- R1, R2: Registers
- MAR: Memory Address Register
- MBR: Memory Buffer Register
- Bus: Data transfer between components
- Mem: Memory operations
- Add, Read, Wait: Memory function types
Register Indirect with Offset
Add to R1 using Register Indirect with Offset with R2 as the index

<table>
<thead>
<tr>
<th>bus → IR</th>
<th>IR addr → bus</th>
<th>result → bus</th>
<th>bus → A LU</th>
<th>bus → PC</th>
<th>bus → R1</th>
<th>bus → R2</th>
<th>bus → M AR</th>
<th>bus → M BR</th>
<th>A LU fun</th>
<th>Mem func</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>add</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>read</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Diagram:
- IR, IR addr → result → ALU, IR
- ALU → op, PC, R1, R2
- R1, R2 → bus
- Bus → MAR, MBR, memory system
- ALU fun
- Mem func
- X indicates a connection or operation.
Register Indirect with Offset
Add to R1 using Register Indirect with Offset with R2 as the index
Pop Instruction

- The pop instruction reads the top of the stack into a register.

Current top of stack

...  Stack pointer, R2

...
Pop Instruction

- Read the memory location whose address is in the stack pointer
Pop Instruction

- Decrement the stack pointer

Current top of stack

...
A pop instruction requires

A. a memory read
B. a memory write
C. both read and write
D. none
Pop R1
Assume R2 is the stack pointer

Read the address in R2 and store R1. Decremented R2.
**Pop R1**  Assume R2 is the stack pointer

*Read the address in R2 and store R1. Decremented R2.*

---

**Diagram:**

- IR → IR addr
- ALU
- PC
- R1
- R2
- MAR
- MBR
- BUS

---

**Table:**

<table>
<thead>
<tr>
<th>bus</th>
<th>IR addr</th>
<th>resu lt</th>
<th>bus</th>
<th>PC</th>
<th>bus</th>
<th>bus</th>
<th>bus</th>
<th>bus</th>
<th>bus</th>
<th>bus</th>
<th>bus</th>
<th>bus</th>
<th>A L U</th>
<th>Mem func</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>dec</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>read</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>wait</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**Legend:**

- **Bus:** IR, PC, R1, R2, ALU, MAR, MBR, Wait
- **Operations:** Dec, Read, Wait
- **Memory System:**"
**Pop R1**  Assume R2 is the stack pointer

*Read the address in R2 and store R1. Decremented R2.*

---

### Architecture Diagram

![Architecture Diagram](image)

---

### Table Representation

<table>
<thead>
<tr>
<th>bus →</th>
<th>IR</th>
<th>IR adr</th>
<th>result</th>
<th>bus</th>
<th>bus</th>
<th>PC</th>
<th>bus</th>
<th>bus</th>
<th>R1</th>
<th>bus</th>
<th>R2</th>
<th>bus</th>
<th>bus</th>
<th>MBR</th>
<th>ALU</th>
<th>Mem func</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>dec</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>read</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**Notes:**

- X: Indicates a connection or operation.
- IR: Instruction Register
- IR addr: Instruction Address
- ALU: Arithmetic Logic Unit
- PC: Program Counter
- R1, R2: Registers
- MAR: Memory Address Register
- MBR: Memory Buffer Register
The ALU action is determined by the

A. Microcode
B. Opcode
C. All of the above
D. None of the above
Function Call

• A function call instruction pushes the return address on the stack and jumps to the start of the function
• The return address is in the Program Counter
CALL `funcaddr` - Push the return address on the stack and jump to the address of the function. R2 is stack ptr

<table>
<thead>
<tr>
<th>bus</th>
<th>IR</th>
<th>result</th>
<th>bus</th>
<th>ALU</th>
<th>bus</th>
<th>PC</th>
<th>bus</th>
<th>R1</th>
<th>bus</th>
<th>R2</th>
<th>bus</th>
<th>MAR</th>
<th>MBR</th>
<th>Mem func</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>inc</td>
</tr>
<tr>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>write</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>wait</td>
</tr>
</tbody>
</table>
CALL funcaddr - Push the return address on the stack and jump to the address of the function. R2 is stack ptr
CALL funcaddr - Push the return address on the stack and jump to the address of the function. R2 is stack ptr
CALL funcaddr - Push the return address on the stack and jump to the address of the function. R2 is stack ptr
Try It

• Write microcode to implement a method return
• Pop the address from the stack and put it in the program counter
return R2 is stack pointer
Register Based Method Call

• Some systems do not use a stack for method calls
• The return address can be saved in a register
• Write the microcode for

  call R2, mymethod
Register Call

Bus

memory system

<table>
<thead>
<tr>
<th>bus</th>
<th>IR</th>
<th>resu</th>
<th>bus</th>
<th>PC</th>
<th>bus</th>
<th>R1</th>
<th>bus</th>
<th>R2</th>
<th>bus</th>
<th>bus</th>
<th>bus</th>
<th>Mem</th>
</tr>
</thead>
<tbody>
<tr>
<td>→ IR</td>
<td>→ IR</td>
<td>→ bus</td>
<td>→ ALU</td>
<td>→ PC</td>
<td>→ R1</td>
<td>→ bus</td>
<td>→ R2</td>
<td>→ MAR</td>
<td>→ MBR</td>
<td>→ A LU</td>
<td>fun</td>
<td>func</td>
</tr>
</tbody>
</table>

X

X

X

X
Return for Register Based Call

• Write the microcode for a return instruction in a register based call system

    return  R2
Memory Operand Instruction

**inc** dog

- The value at the memory location (dog) is incremented by one.
- Direct memory addressing
- Calculations occur in the CPU
  - Read memory value
  - Increment number
  - Store value back to memory
### inc dog

![Diagram showing the components of a computer system with data flow between them.]

#### Diagram:
- **Bus**: The main horizontal bar represents the bus system where data flows.
- **MAR**: Memory Address Register.
- **MBR**: Memory Buffer Register.
- **ALU**: Arithmetic Logic Unit.
- **PC**: Program Counter.
- **R1** and **R2**: Registers.
- **IR**: Instruction Register.
- **Result**: The output of the ALU.

#### Table:
<table>
<thead>
<tr>
<th>bus → IR</th>
<th>IR addr bus</th>
<th>result bus</th>
<th>ALU</th>
<th>bus → PC</th>
<th>bus → R1</th>
<th>bus → R2</th>
<th>bus → MAR</th>
<th>bus → MBR</th>
<th>M A L U</th>
<th>Mem func</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>inc</td>
</tr>
<tr>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Notes:
- The table shows the transitions of data between different components of the computer system.
- The rows and columns indicate the direction of data flow.
- The 'x' symbols represent connections between the components.
- The 'Mem func' column indicates memory function operations like 'read', 'wait', 'inc', 'write', and 'wait'.
### inc dog

![Diagram showing the processing of an instruction](image)

#### Table: Instruction Processing

<table>
<thead>
<tr>
<th></th>
<th>IR addr</th>
<th>IR</th>
<th>result</th>
<th>bus to ALU</th>
<th>bus to PC</th>
<th>bus to R1</th>
<th>bus to R2</th>
<th>bus to MAR</th>
<th>bus to MBR</th>
<th>ALU fun</th>
<th>Mem func</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>read</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>wait</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>inc</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>write</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>wait</td>
<td></td>
</tr>
</tbody>
</table>
## inc dog

![Diagram of a computer system showing the flow of data and control signals through the bus, IR, ALU, PC, R1, R2, MAR, MBR, and memory system.](image)

### Table: Instruction Execution Flow

<table>
<thead>
<tr>
<th>Bus Flow</th>
<th>IR Address</th>
<th>Result</th>
<th>ALU</th>
<th>Bus to ALU</th>
<th>PC</th>
<th>Bus to PC</th>
<th>Bus to R1</th>
<th>Bus to R2</th>
<th>Bus to MAR</th>
<th>Bus to MBR</th>
<th>MBR to Bus</th>
<th>ALU Function</th>
<th>Memory Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>bus → IR</td>
<td>IR adr → bus</td>
<td>result → bus</td>
<td>ALU</td>
<td>bus → opr nd</td>
<td>PC</td>
<td>bus → PC</td>
<td>bus → R1</td>
<td>bus → R2</td>
<td>bus → MAR</td>
<td>bus → MBR</td>
<td>MBR to bus</td>
<td>ALU fun</td>
<td>Mem func</td>
</tr>
<tr>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>read</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>wait</td>
</tr>
<tr>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>inc</td>
</tr>
<tr>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>write</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>wait</td>
</tr>
</tbody>
</table>
Microcode Beyond the Simple Computer

• Next instruction field
  – Normally points to the next line
  – May point to fetch of next instruction

• ALU function taken from instruction register opcode

• Register selection from the instruction

• Status flags or comparison mechanism
Microphobia

An unreasoning fear of microcode
Fetch and PC Increment

• The fetch/execute cycle always starts with reading the instruction from memory and incrementing the program counter

• In our simple computer, this takes three lines of microcode

• This is the only time anything is loaded into the instruction register
Jump Instructions

- The last microcode step of a jump is almost always copies a value into the program counter
- Jump instructions rarely access memory unless they are pushing or popping something on the stack
Arithmetic

• Most arithmetic instructions have the steps
  – Copy something into the operand reg
  – Put a value on the bus and do it
  – Copy the result register someplace

• In a more realistic system, the ALU function would be determined by the opcode of the instruction