### Table and Switches

<table>
<thead>
<tr>
<th>bus → IR</th>
<th>IR addr → bus</th>
<th>A</th>
<th>ALU opnd → result</th>
<th>bus → PC</th>
<th>PC → bus</th>
<th>R1 → bus</th>
<th>R2 → bus</th>
<th>bus → MAR</th>
<th>MBR → bus</th>
<th>ALU func</th>
<th>Mem func</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

### Memory Read

- Copy the address to read into the Memory Address Register.
- Tell the memory system to “READ”. This only starts the read which will take several cycles.
- A following microcode instruction must “WAIT” until the read is complete.
- Copy the data from the Memory Buffer Register to the desired register or ALU.
Memory Read

- Copy the address to be written into the Memory Address Register.
- Copy the data into the Memory Buffer Register.
- Tell the memory system to “WRITE”. This only starts the write which will take several cycles.
- A future microcode instruction must “WAIT” before doing another memory function.

Memory Write

- Copy the address to be written into the Memory Address Register.
- Copy the data into the Memory Buffer Register.
- Tell the memory system to “WRITE”. This only starts the write which will take several cycles.
- A future microcode instruction must “WAIT” before doing another memory function.
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 somewhere
- In a more realistic system, the ALU function would be determined by the opcode of the instruction.
## What does this microcode implement?

<table>
<thead>
<tr>
<th>bus → IR</th>
<th>IR addr → bus</th>
<th>resu rt → bus</th>
<th>A, L, U, opr nd</th>
<th>bus → PC</th>
<th>bus → R1</th>
<th>R1 → bus</th>
<th>R2 → bus</th>
<th>bus → MAB</th>
<th>MAB R → bus</th>
<th>ALU func</th>
<th>Mem func</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

1. push
2. pop
3. add
4. jump

## Stack Instructions

- The stack is kept in memory
- We assume R2 points to the top of the stack

### Pop Instruction

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

### Pop Instruction

- Decrement the stack pointer
Push Instruction

- Increment the stack pointer

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.
- We will assume register R2 is the stack pointer

Increment the stack pointer, R2
Put the new top of stack address back in R2 and set the memory address to the top of stack

Write the return address, PC, to the top of stack in memory

Jump to the address of the function

Try It

- Write the microcode to implement a return instruction
- Pop the return address off the stack
1. Read the top of stack
2. Decrement the stack pointer
3. Put the value read in the Program Counter