Still
More on Microcode

COMP375 Computer Architecture and Organization

Addressing modes

- immediate
- register
- memory direct
- register indirect
- register indirect with offset
- memory indirect
- register + offset memory indirect
- displacement

Immediate

- The data is part of the instruction.
- Immediate data items are read-only.
- There is usually a size limit.

Register

- The data is in a CPU register.
- The instruction *might* indicate which register

\[ \text{Instruction} \downarrow \text{data register} \]

\[ \text{address} \]

\[ \text{data} \]

\[ \text{memory} \]
Memory Direct

- The data is in memory.
- The instruction contains the address of the memory location.

Register Indirect

- The address of the data is in a CPU register.
- Useful if the address is calculated.

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.

Memory Indirect

- A memory location contains the address of the data.
- Useful for pointers.
Register Offset & Memory Indirect
• The sum of the instruction offset field and a register value gives the location of the address in memory.
• Useful when addressing ref parameters.

Displacement
• The address of the data is the sum of the instruction offset field and the program counter.
• Used for short jumps

Stack Addressing
• The same as Register Indirect with Offset using the stack pointer register
• Useful when addressing local variables or parameters.

The fastest addressing mode in this list is
1. Immediate
2. Direct
3. Memory Indirect
4. Register Indirect with Offset
Microcode Steps

- A microcode step is executed each clock cycle.
- For each microcode step, the next line of the microcode store is read.
- The microcode data determines the state of the switches connecting registers to the bus.
**Microcode Programs**

- The sequence of switch settings in the microcode store is a microcode program.
- Real microcode programs can conditionally jump to another line in the program.
- Often each line of the microcode contains a field with the address of the next microcode step.

**Jump Instruction**

Jump to the address given in the instruction

*Direct Addressing*

<table>
<thead>
<tr>
<th>bus</th>
<th>IR</th>
<th>resu</th>
<th>A</th>
<th>L</th>
<th>U</th>
<th>opr</th>
<th>PC</th>
<th>bus</th>
<th>R1</th>
<th>bus</th>
<th>R2</th>
<th>bus</th>
<th>MA</th>
<th>M</th>
<th>B</th>
<th>R</th>
<th>bus</th>
<th>ALU</th>
<th>Mem</th>
<th>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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Opcode Determines Microcode Steps**

- After the microcode steps to fetch the instruction and increment the program counter, the next microcode step executed depends on the opcode of the instruction.

- Most of our microcode will ignore the opcode and just concentrate on one instruction at a time.

**Opcode and Microcode Step**
If each microcode instruction instruction contains the address of the next instruction, this can best be modeled by a:

1. Sequential array
2. Binary Tree
3. Linked List
4. Quad Tree
5. Heap
6. Data Structures doesn’t apply to architecture

**Internal CPU Bus**

- Only one register can put its value on the bus at a time.
- The value on the bus can be copied into many registers at the same time.

**Optimizing Microcode**

- The more microcode steps required, the longer an instruction will take to execute.
- The value on the bus can be copied into multiple registers.
- Instead of copying A to B then B to C, the value in A can be copied to both B and C.
- Unrelated microcode steps can be executed before waiting for a memory access to complete.

**Instruction Fetch & Program Counter Increment**
**Instruction Fetch & Program Counter Increment**

Combining steps to reduce total time

<table>
<thead>
<tr>
<th>bus → IR</th>
<th>IR addr → bus</th>
<th>result → bus</th>
<th>ALU → result</th>
<th>bus → ALU fun</th>
<th>ALU → bus</th>
<th>PC → bus</th>
<th>PC → R1</th>
<th>R1 → bus</th>
<th>bus → R2</th>
<th>R2 → bus</th>
<th>bus → MA</th>
<th>bus → MBR</th>
<th>MBR → bus</th>
<th>Mem func</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X inc</td>
<td>X</td>
<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>X</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>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>
</tr>
</tbody>
</table>

**Remember!**

- The three lines of microcode required to fetch an instruction and increment the program counter are common to all instructions.
- These three lines will appear on the quiz on Friday and on the next exam and maybe even the final exam.

**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

**Write Some Microcode**

Using the worksheet provided, write the microcode to:

- **Subtract R1, cow** memory direct
- **Do not bother to show the instruction fetch and program counter increment steps.**
Write Some Microcode

Using the worksheet provided, write the microcode to:

• **Jump with Program Counter Displacement**

• Jump to the address that is the sum of the address field and the PC

  • Do not bother to show the instruction fetch and program counter increment steps.
Function Calls

- To call a function or method, the program counter is pushed on the stack and then the program counter is loaded with the address of the function.
- This puts the address of the instruction after the function call on the stack.
- To return the return address is popped from the stack and loaded into the program counter.

Write Some Microcode

Using the worksheet provided, write the microcode to:

- **Call Instruction** - Push the return address on the stack and jump to the address of the function. Assume R2 is stack pointer.

- Increment R2 and put the value in the MAR. Write the PC to that address and then load the PC from the instruction address field.

  - *Do not bother to show the instruction fetch and program counter increment steps.*

Call someplace

*R2 is the stack pointer*

<table>
<thead>
<tr>
<th>bus → IR</th>
<th>IR addr → bus</th>
<th>reslt → IR</th>
<th>A L U oprnd</th>
<th>bus → PC</th>
<th>bus → bus</th>
<th>bus → bus</th>
<th>bus → R2</th>
<th>bus → bus</th>
<th>bus → MAR</th>
<th>bus → M BR</th>
<th>bus → bus</th>
<th>A L U func</th>
<th>Mem func</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>inc</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>X</td>
<td>inc</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>X</td>
<td>inc</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>X</td>
<td>inc</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Interrupts

- Read only registers contain the fixed address of special memory locations.
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.

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.