Lab 4

Due date: Friday, December 5th
CSE 140L
Fall 2003
Datapath

decoder

R0
R1
R2
R3

mux

load

mem_in

mem_out

w_addr

we

mux

r_addr1

mux

r_addr2

ALU

alu_op

zero_flag

mem_out

mem_in

load
• Instruction format: <opcode, operand1, operand2>
• Instructions:
  000 -- --  noop
  001 aa --  set    R[aa] = 1
  010 aa --  increment    R[aa] = R[aa] + 1
  011 aa --  decrement    R[aa] = R[aa] - 1
  100 aa --  load    R[aa] = mem_in
  101 aa --  store    mem_out = R[aa]
  110 aa bb  add    R[aa] = R[aa] + R[bb]
  111 aa bb  copy    R[aa] = R[bb]
## Nano-decoder

<table>
<thead>
<tr>
<th>opcode</th>
<th>opcode</th>
<th>opr1</th>
<th>opr2</th>
<th>alu_op</th>
<th>raddr1</th>
<th>raddr2</th>
<th>waddr</th>
<th>we</th>
<th>load</th>
</tr>
</thead>
<tbody>
<tr>
<td>noop</td>
<td>000</td>
<td>--</td>
<td>--</td>
<td>000</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>set</td>
<td>001</td>
<td>aa</td>
<td>--</td>
<td>001</td>
<td>--</td>
<td>--</td>
<td>aa</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>incr</td>
<td>010</td>
<td>aa</td>
<td>--</td>
<td>010</td>
<td>aa</td>
<td>--</td>
<td>aa</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>decre</td>
<td>011</td>
<td>aa</td>
<td>--</td>
<td>011</td>
<td>aa</td>
<td>--</td>
<td>aa</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>load</td>
<td>100</td>
<td>aa</td>
<td>--</td>
<td>100</td>
<td>--</td>
<td>--</td>
<td>aa</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>store</td>
<td>101</td>
<td>aa</td>
<td>--</td>
<td>101</td>
<td>aa</td>
<td>--</td>
<td>--</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>add</td>
<td>110</td>
<td>aa</td>
<td>bb</td>
<td>110</td>
<td>aa</td>
<td>bb</td>
<td>aa</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>copy</td>
<td>111</td>
<td>aa</td>
<td>bb</td>
<td>111</td>
<td>bb</td>
<td>--</td>
<td>aa</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

- Hints: can assign don’t cares so that alu_op = opcode, and raddr2 = opr2
- waddr (write address), we (write enable), load, and raddr1 logic more complicated
More on Datapath

- Assume 4-bits represent only positive numbers 0 ... 15

- Increment can be implemented as
  - \( A + 0, \text{carry}_\text{in} = 1 \)

- Decrement can be implemented as
  - \( A + "1111", \text{carry}_\text{in} = 0 \)

- \text{zero}_\text{flag} = 1 \) if the output of the ALU is “0000”
Fibonacci Sequence

- Fibonacci sequence
  - $F(1) = 1$
  - $F(2) = 1$
  - $F(N) = F(N-1) + F(N-2)$
  - e.g. 1, 1, 2, 3, 5, 8, 13 ...
  - With a 4-bit datapath, can only count up to 15, or assume $N \leq 7$, $F(7) = 13$

Pseudo code

```cpp
int fibonacci (int N)
{
    int N1 = 1, N2 = 1;
    int F, temp, c;
    for (c = N-2; c > 0; c--)
    {
        temp = N1;
        N1 = N1 + N2;
        N2 = temp;
    }
    F = N1;
    return F;
}
```
Pseudo Machine Code

• Specification
  - Wait until start = 1
  - Assume N-2 is provided at mem_in for computing F(N)
  - Use store instruction to output the final answer F(N) to mem_out and set done = 1 (done = 0 during other cycles)

• Register allocation
  - R0: count
  - R1: N1
  - R2: N2
  - R3: temp

• Pseudo machine code
  while (not start) {
    noop
  }
  load R0 // set c = N-2
  set R1 // set N1 = 1
  set R2 // set N2 = 1
  store R0 // test zero
  while (not zero_flag) {
    copy R3 R1
    add R1 R2
    copy R2 R3
    decre R0
  }
  store R1, done = 1
  go back to initial state
# FSM (Moore Machine)

<table>
<thead>
<tr>
<th>state</th>
<th>start/zero_flag</th>
<th>opcode</th>
<th>opr1</th>
<th>opr2</th>
<th>done</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A A B</td>
<td>100</td>
<td>R0</td>
<td>--</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>D D B</td>
<td>001</td>
<td>R1</td>
<td>--</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>E E B</td>
<td>001</td>
<td>R2</td>
<td>--</td>
<td>0</td>
</tr>
<tr>
<td>E</td>
<td>F J B</td>
<td>101</td>
<td>R0</td>
<td>--</td>
<td>0</td>
</tr>
<tr>
<td>F</td>
<td>G G B</td>
<td>111</td>
<td>R3</td>
<td>R1</td>
<td>0</td>
</tr>
<tr>
<td>G</td>
<td>H H B</td>
<td>110</td>
<td>R1</td>
<td>R2</td>
<td>0</td>
</tr>
<tr>
<td>H</td>
<td>I I B</td>
<td>111</td>
<td>R2</td>
<td>R3</td>
<td>0</td>
</tr>
<tr>
<td>I</td>
<td>F J B</td>
<td>011</td>
<td>R0</td>
<td>--</td>
<td>0</td>
</tr>
<tr>
<td>J</td>
<td>A A B</td>
<td>101</td>
<td>R1</td>
<td>--</td>
<td>1</td>
</tr>
</tbody>
</table>

A: while (not start) {
    noop
}
B: load R0
C: set R1
D: set R2
E: store R0
    while (not zero_flag) {
        F: copy R3 R1
        G: add R1 R2
        H: copy R2 R3
        I: decr R0
    }
J: store R1, done = 1
    go back to initial state

* Note: when start = 1, just start the calculation over
Grading

• Part I (12 out of 36 points)
• Part II (12 out of 36 points)
• Part III (12 out of 36 points)
• Total 36 points
• 40% of overall grade
Part I

• Part I (12 out of 36 points)
  - Test the ALU to make sure that the combinational logic works for these alu_ops (set, incr, decr, store, add, copy)
  - For single operand operations, assume operand provided at operand1 (left arm of ALU)

Sample test cases

1. set (just set output to 1)
2. incr 4
3. dec 1
4. store (just pass operand1 to output)
5. add 2 + 4
6. copy (just pass operand1 to output)

• 2 points for each of the 6 test cases.

• Actual values for test cases during live demo will change
Part II

- Part II (12 out of 36 points)
  - Test entire datapath and nano-decoder by running through sequences of instructions.
  - You'll be given two sequences
  - Each sequence is worth 6 points.
  - In each sequence, there will be 3 places where `mem_out` will be checked and 1 place where `zero_flag` will be checked, a total of 4 checks, each worth 1.5 points (4 x 1.5 = 6)
  - Actual test sequences will change at live demo

• Sample sequence I
  - set R1 // R1 = 1
  - set R2 // R2 = 1
  - add R1 R2 // R1 = 2
  - store R1 // mem_out = R1
  - add R1 R2 // R1 = 3
  - decr R2 // R2 = 0
    // zero_flag = 1
  - store R1 // mem_out = R1
  - store R2 // mem_out = R2

• Sample sequence II
  - load R0 // R0 = 7 (mem_in)
  - load R3 // R3 = 5 (mem_in)
  - store R3 // mem_out = R3
  - add R3 R0 // R3 = 12
  - decr R0 // R0 = 6
    // zero_flag = 0
  - incr R3 // R3 = 13
  - store R1 // mem_out = R0
  - store R2 // mem_out = R3
Part III

- Part III (12 out of 36 points)
  - Test to see if the fibonacci calculator works
  - N-2 is provided for calculating F(N)
  - N will be less than or equal to 7, which means N-2 provided at mem_in will be less than or equal to 5
  - Two fibonacci calculations will be tested, each worth 6 points
  - For each test, either it works or it doesn’t, i.e. 0 or 6 points

- Sample case I
  - Provided with N-2 = 5
  - Calculate F(N) = F(7) = 13

- Sample case II
  - Provided with N-2 = 1
  - Calculate F(N) = F(3) = 2

- Assume
  - N-2 >= 0
  - Implies N >= 2