Execution of Add (R3), R1

- Fetch the instruction
- Fetch the first operand (R3)
- Perform the addition
- Load result to R1

What should MDRin be?

Execution

- Step 1: PC loaded into MAR, read request to memory, MUX gives 4, added to B (PC) and stored in Z
- Step 2: Z moved to PC while waiting for memory
- Step 3: Word fetched from memory and loaded into IR
- Step 4: figure out what the instruction should do and set control circuitry for steps 4-7. R3 transferred to MAR and memory read operation initiated
- Step 5: contents of R1 moved to Y
- Step 6: read operation completed and is in MDR as well as B input of ALU. Select Y as second input of ALU and add performed
- Step 7: result is transferred to R1, End causes a goto step 1

Which steps are the instruction fetch?

Branch Instructions

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PCout, MAR in, Read, Select4, Add, Zm</td>
</tr>
<tr>
<td>2</td>
<td>Zout, PCout, Ym, WMFC</td>
</tr>
<tr>
<td>3</td>
<td>MDRin, IRin</td>
</tr>
<tr>
<td>4</td>
<td>Offset-field-of-IRout, Add, Zm</td>
</tr>
<tr>
<td>5</td>
<td>Zout, PCout, End</td>
</tr>
</tbody>
</table>

Now do you understand why the branch offset is calculated from the next address to be executed?

For conditional e.g. branch < 0, step 4 is replaced with Offset-field-of-IRout, Add, Zin, if N=0 then End

Multiple Buses

- One disadvantage of our single bus scheme is that only one data item can be transferred over the bus per cycle
- A solution is multiple internal buses
- All registers combined into a register file with 3 ports
  - Why are there 2 outputs?
  - What is the input for?
  - What does 3 port mean?
- Buses A and B allow simultaneous transfer of the two operands for the ALU
  - ALU is able to just pass one of its operands to R e.g. R=X
- Incrementer unit computes PC+4, means we don’t need the ALU for this
  - ALU still has a 4 input for other instructions such as postincrement
Three bus datapath

- What does this do?
- (WMFC means wait for memory function completed)
- What are the advantages and disadvantages over a single bus?

Step Action

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PCout, R=B, MARin, Read, IncPC</td>
</tr>
<tr>
<td>2</td>
<td>WMFC</td>
</tr>
<tr>
<td>3</td>
<td>MDRout, R=B, IRin</td>
</tr>
<tr>
<td>4</td>
<td>R4outA, R5outB, SelectA, Add, R6in, End</td>
</tr>
</tbody>
</table>

Hardwired Control

- How do we generate the control signals?
  - Hardwired control
  - Microprogrammed control
- A hardwired control is called a finite state machine
  - Sequences using a counter and produces control signals at the right time
  - Control signals are functions of the IR, external inputs and condition codes
  - Can you give an example for each?

Microprogrammed Control

- The control signals are stored in a memory as sequences of control words which are the individual bits of the control signals
- Microinstructions are executed in a manner similar to machine code

<table>
<thead>
<tr>
<th>Microinstruction</th>
<th>IR</th>
<th>MAR</th>
<th>MAR</th>
<th>IR</th>
<th>Y</th>
<th>M</th>
<th>X</th>
<th>Add</th>
<th>Z</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
<th>WMFC</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Microprogrammed Control Unit

Address Microinstruction

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>PCout, MARin, Read, Select, Add, Zout</td>
</tr>
<tr>
<td>1</td>
<td>Zout, PCout, Yin, WMFC</td>
</tr>
<tr>
<td>2</td>
<td>MDRout, R=B, IRin</td>
</tr>
<tr>
<td>3</td>
<td>Branch to starting address of appropriate microroute</td>
</tr>
<tr>
<td>4</td>
<td>If N=0, then branch to microinstruction 0</td>
</tr>
<tr>
<td>5</td>
<td>Offset-field of IR out, SelectY, Add, Zout</td>
</tr>
<tr>
<td>6</td>
<td>Zout, PCout, End</td>
</tr>
</tbody>
</table>

Scheme to allow Conditional Branching

- "Starting and branch address generator"
  - Loads new address into uPC when instructed
  - Has condition codes and external inputs which can affect uPC
- uPC incremented every cycle except
  - When new instruction loaded into IR, uPC loaded with starting address of the microroutine
  - For taken branches, uPC updated to branch address
  - For End microinstruction, uPC set to 0
Floating Point

Fractions

- To date we have only considered integers
- We can also assume binary fractions
  \[ B = b_0.b_{-1}b_{-2}...b_{-(n-1)} \]
  - Value is \( F(B) = -b_0 \times 2^0 + b_{-1} \times 2^{-1} + ... + b_{-(n-1)} \times 2^{-(n-1)} \)
  - Range of \( F \) is \(-1 \leq F \leq 1-2^{-(n-1)}\)

- Range of 32-bit numbers
  - Signed integer approx: \( 0 \) to \( \pm 2.15 \times 10^9 \)
  - Signed fraction approx: \( \pm 4.55 \times 10^{-10} \) to \( \pm 1 \)

- Some commonly used numbers
  - Avogadro's number \( 6.0247 \times 10^{23} \)
  - Planck's constant \( 6.6254 \times 10^{-27} \)

- Need a way to adjust position of the binary point as the computation proceeds c.f. scientific notation
  - Floating point numbers

Binary Fraction to Decimal Fraction

- Example: What is the binary value 0.011010 in decimal?
  \[ \frac{13}{32} = 0.40625 \]

- Example: What is 0.110 in binary?
  \[ 0.1 \times 2 = 0 \]
  \[ 0.2 \times 2 = 0 \]
  \[ 0.4 \times 2 = 0 \]
  \[ 0.8 \times 2 = 1 \]
  \[ 0.6 \times 2 = 1.2 \]
  \[ 0.2 \times 2 = 0.4 \] and then repeating 0.4, 0.8, 0.6

- Answer: 0.0011 0011 0011 0011 0011 0011 0011 .....
**Floating point number**

- Much as for scientific notation as in $6.6254 \times 10^{-27}$, we use
  - A normalised number of certain accuracy e.g. 6.6254 is called the **mantissa**
  - Scale factors to determine the position of the decimal point e.g. $10^{-27}$ indicates position of decimal point and is called the **exponent** (the base is implied)
  - Sign bit

**Normalised Numbers**

- Floating Point Numbers can have multiple forms, e.g.
  - $0.232 \times 10^4 = 2.32 \times 10^3$
  - $23.2 \times 10^0$
  - $232 \times 10^{-2}$

- It’s desirable for each number to have a unique representation => **Normalised Form**

- We’ll normalise Mantissa’s in the Range $[1 .. R)$ where $R$ is the Base, e.g. $[1 .. 2)$ for BINARY