Announcements

- Reading Assignment – Read Chapter 5, Section 5.4 – end of chapter
- Lab 2 – Postlab
- Homework clarifications – 4.11, 4.12, 4.19

Microprogramming

CS 333
Fall 2006

Microprogramming Motivation

- Control functions
  - Complex
  - Manageable for smaller, more regular instruction sets, but consider:
    - IA-32

IA-32

- Intel Architecture 32-bit
  - CISC-like instruction set architecture
    - Several hundred instructions of varying classes
    - Varying length instructions
    - Many addressing modes
    - First implementation 80386
    - Pentium, … Pentium 4, …AMD K7

Example IA-32 add Instruction Formats

- Varying lengths/formats, makes control unit design more complicated

Microprogramming!

- Thousands of states
  - Hundreds of different control sequences
- Control signals can be thought of as if they are instructions executed by the datapath
- Simplifies control design
Microprogramming Process
1. Define microinstruction format
2. Create microprogram
3. Implement the microprogram

Defining the Microinstruction Format

Microinstructions
- To avoid confusion with ISA (instruction set architecture)
  - microinstructions
    - Defines set of datapath control signals to assert in a given state
    - Executing a microinstruction
      - Asserts control signals specified in microinstruction

Microprogram
- Algorithmic description for how to execute instructions (from ISA) in a program by asserting a sequence of control signals
- Each instruction in ISA has a list of microinstructions
  - Ex., control sequences in concrete RTN for SRC

Sequencing
- The next microinstruction to be executed needs to be specified
  - Analogy
    - In programs we use functions/methods to reuse commonly executed sequences of instructions (control flow/subroutines)
    - Microprograms are similar
      - Several instructions in the ISA may have similar control sequences (these can be reused instead of recoded)
      - Example, every instruction at least needs to be fetched before execution

Microprogramming
- Designing the control symbolically
  - Analogy:
    - Assembly language is a symbolic representation of machine instructions
      - Fields: op code, registers, offsets, immediate field, etc.
    - Microprogram is a symbolic representation of microinstructions
      - Also has fields. Many different arrangements
      - Microcoded control units are used to implement complex microprograms
Microinstruction Format

• Need to choose:
  – Number of fields
  – What control signals are specified by each field
• Would like:
  – Readability:
    • Format needs to be simple enough to write and understand the microprogram
  – Difficult to write inconsistent microinstructions
    • Don’t want a particular control signal to be specified as two different values in one microinstruction
    • Signals never asserted simultaneously can share the same field

More Parallels with Assembly Programs

• Microinstruction fields may allow combinations that can’t be supported by the datapath
  – microassembler – checks/flags these errors

Control Store

• Often a PLA (programmable logic array) or ROM (read-only memory)
  – Each entry has an address

- Ways of Sequencing

  1. Increment address of current microinstruction
     • like sequential execution of instructions
     • often is default
  2. Branch to the next microinstruction that executes the next instruction (ISA)
     • Branch to the instruction fetch microinstruction sequence
  3. Dispatch – Lookup table (PLA)

Creating a Microprogram

Example Microinstruction Fields

• ALU control
  – Add – cause ALU to add
  – Subtract – cause ALU to subtract
• SRC1 – first source register to ALU
  – PC
  – Register A – first ALU input
• SRC2 – second operand
  – Register B, second ALU input
  – 4- for PC + 4
  – Extend (sign extend)
Example Microinstruction Fields

- Register Control
  - Read – read rb, rc
  - Write ALU – write ALU output to ra in register file
- Memory
  - Read PC – read memory using PC as address; write result to IR
  - Read ALU – read from memory using ALU output as address; write result to MD
  - Write ALU – write to memory ALU output as address; write result to MD

More Microinstruction Fields

- PCWrite control
  - ALU – write output of ALU into PC
  - Branch address – write PC with branch target from instruction
- Sequencing
  - Seq – choose next instruction sequentially
  - Fetch – go to first microinstruction to begin a new instruction
  - Dispatch – dispatch using ROM

Instruction Fetch Microinstructions

<table>
<thead>
<tr>
<th>Label</th>
<th>ALU Ctrl</th>
<th>SRC 1</th>
<th>SRC 2</th>
<th>Reg Ctrl</th>
<th>Mem</th>
<th>PC Write</th>
<th>Seq</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fetch</td>
<td>Add</td>
<td>PC</td>
<td>4</td>
<td>Read PC</td>
<td>ALU</td>
<td>Seq</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Add</td>
<td>PC</td>
<td>Exts hft</td>
<td>Read</td>
<td></td>
<td>Dispatch</td>
<td></td>
</tr>
</tbody>
</table>

Memory Reference Instructions

<table>
<thead>
<tr>
<th>Label</th>
<th>ALU Ctrl</th>
<th>SRC 1</th>
<th>SRC 2</th>
<th>Reg Ctrl</th>
<th>Mem</th>
<th>PC Write</th>
<th>Seq</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mem</td>
<td>Add</td>
<td>A</td>
<td>Extend</td>
<td>Read PC</td>
<td></td>
<td>Seq</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Write ALU</td>
<td></td>
<td>Fetch</td>
<td></td>
</tr>
<tr>
<td>SW</td>
<td></td>
<td></td>
<td></td>
<td>Write ALU</td>
<td></td>
<td>Fetch</td>
<td></td>
</tr>
</tbody>
</table>

Add Instruction

<table>
<thead>
<tr>
<th>Label</th>
<th>ALU Ctrl</th>
<th>SRC 1</th>
<th>SRC 2</th>
<th>Reg Ctrl</th>
<th>Mem</th>
<th>PC Write</th>
<th>Seq</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddI nst</td>
<td>Add</td>
<td>A</td>
<td>B</td>
<td></td>
<td></td>
<td>Seq</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Write ALU</td>
<td></td>
<td>Fetch</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Branch Instruction (brzr)

<table>
<thead>
<tr>
<th>Label</th>
<th>ALU Ctrl</th>
<th>SRC 1</th>
<th>SRC 2</th>
<th>Reg Ctrl</th>
<th>Mem</th>
<th>PC Write</th>
<th>Seq</th>
</tr>
</thead>
<tbody>
<tr>
<td>brzr</td>
<td>Subt</td>
<td>A</td>
<td>B</td>
<td>ALUout</td>
<td></td>
<td>Fetch</td>
<td></td>
</tr>
</tbody>
</table>
The Final Microprogram

Implementing the Microprogram

Translating Microprogram to Hardware

- Need:
  1. Sequencing function
  2. Method of storing main control function

Comparisons

Assembly language program
Microprogram

Assembler
µAssembler

Machine Language
Microcode

...00010001...

...000011100...

Copyright © 2004 Pearson Prentice Hall, Inc.
Comparisons (cont’d)

Machine Language/Program store (RAM) 1024

Microcode/Control store (ROM/PLA) 16

Different Microcode Schemes

1. Mostly Horizontal (Example)
2. Horizontal
3. Vertical

Horizontal

- 1 bit for every control signal
  - textbook example

Microinstruction format

<table>
<thead>
<tr>
<th>Branch control</th>
<th>Control signals</th>
<th>Branch address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 _ _ _ _ _ _ _</td>
<td>_ _ _ _ _ _ _</td>
<td>_ _ _ _ _ _ _</td>
</tr>
<tr>
<td>_ _ _ _ _ _ _ _ _ _ _</td>
<td>_ _ _ _ _ _ _ _ _ _ _</td>
<td>_ _ _ _ _ _ _ _ _ _ _</td>
</tr>
</tbody>
</table>

Vertical

- Each distinct microinstruction is a single signal sent to control points to be asserted

Horizontal vs. Vertical Tradeoffs

- Horizontal
  - wider microinstruction word width
    - 1 bit for each control signal
- Vertical
  - narrower microinstruction word width
    - Example, 512 distinct microinstructions
      - 512=2^9, so 9 bits needed to encode control word
      - control signals are fanned out to control points in datapath
      - can be impractical, but saves bit count
Microprogramming Tradeoffs

Pros of Microprogramming
- Ease of design
- Flexibility
- Easy to adapt to changes in organization, timing, technology
- Can make changes late in design cycle, or even in the field
- Can implement very powerful instruction sets (just more control memory)
- Generality
- Can implement multiple instruction sets on same machine.
- Can tailor instruction set to application.
- Compatibility
- Many organizations, same instruction set

Cons of Microprogramming
- Costly to implement
- Slow

Then vs. Now
- Control units are now integral part of the processor (same die)
  – Can’t be changed independent of processor
- ROM no longer faster than RAM
- Instruction sets are smaller
  – Reduced complexity
- CAD tools are better now
  – Relative difficulty of implementing control logic vs. ROM/PLA is small

Things to Think About

Question 1
- Is adding a complex instruction implemented with microprogramming faster than using a sequence of simpler instructions?
Example: IA-32

- LOOP
  - Decrements register and branches to a label if the decremented register is not zero
  - Used for loops which have a fixed number of iterations
  **Slower than the macrocode sequence of simpler instructions**
Optimizing compilers avoid generating LOOP instructions

Question 2

- If there’s space in the control store, why not implement new and cool instructions?

  **Upward compatibility** – future models will need to support the instruction, even if the space is needed later

Example: Intel 80286

- Many instructions added to the instruction set
  - Protection mechanism (mostly unused today)
    - Still has to be implemented in newer implementations
  - Decimal instructions (decimal arithmetic on bytes)
    - Rarely used today, performing binary arithmetic and converting to and from decimal is faster
    - Still has to be implemented in newer implementations

Microprogramming Summary

- Why?
- Microprogramming process
  - Design microinstruction format
  - Write the microprogram
  - Implement the microprogram
- Terminology
  - microinstruction, microprogram, microcode

Microprogramming Summary (cont’d)

- Similarities and differences between assembly language programs and microprograms
- Difference between microprogram and microcode
- Different microcode schemes
  - Horizontal
  - Vertical
- Microprogramming tradeoffs