Lecture 24: Sample MicroArchitecture

- Control Unit
- ALU
- Registers
- Instructions
- Instruction Decoding
- MicroInstructions

About the Sample MicroArchitecture

- We’ll be using a Sample Microarchitecture to learn about microprogramming.
- It’s described in a handout on the web. This was generated by Prof. Hamel from a number of different sources (listed in the handout)
- Why not use Intel?
  - think back on how complicated the machine code was.
  - microcode to implement all those instructions would be too much to learn in three days!

Control Unit

- Control unit: works with the microprogram to interpret microinstructions.
- MAP – mapping memory used to get the start address in our microprogram.
- Control memory (Control store) – holds our microprogram
- MUX conditional codes – status bits from the ALU
  - N – set if ALU output is negative
  - Z – set if ALU output is zero

ALU

- ALU has three components:
  - general registers
  - ALU itself
  - Shift unit
- ALU Functions:
  - addition (+)
  - binary AND (and)
  - complement left operand (com)
  - pass left operand through unchanged (default)
ALU, cont.

- Shift functions:
  - left shift (lshift)
  - right shift (rshift)
  - no shift (default)
- A and B latches are used to present stable data to the ALU
- Status bits are N and Z (described earlier)

Internal Registers

- We’ve seen registers that the programmer has access to (for Intel: AX, BX, …).
- Units within the processor (such as the control unit or ALU) have their own internal registers.

MicroArchitecture Registers

- General Registers:
  - ACC (Register 0): Accumulator
  - PC (Register 1): Program counter
  - IR (Register 2): Instruction register
  - TMP (Register 3): Temporary register, used for instruction encoding and other things
  - AMASK (Register 4): Address mask, used to mask out the high-order bits of an instruction (leaving 13 address bits)
  - 1 (Register 5): Contains the constant 1

MicroArchitecture Registers

- Other Registers:
  - MAR: Memory address register, holds the address of an operand (13 bits wide)
  - MBR: Memory buffer register, holds the item read from memory
  - MIR: Microprogram instruction register – holds the current microinstruction
  - MPC: Microprogram program counter – points to the next microinstruction
MicroInstructions

- Microinstructions contain many fields, each controlling a different unit.
  - Commands (such as RD, WR) – set to one to send a signal on the respective control lines.
  - Function codes (such as ALU, SH) – represent which operation is required by the ALU or shifter.
  - Addresses (such as A, or B) – give the register number that should be operated on.

Instruction Set

- To keep things simple, we’ll be using a simplified machine language (NOT Intel!)
- In this machine language instructions are 16 bits:
  - high-order 3 bits are the opcode
  - low-order 13 bits are the address

Instruction Set, cont.

- Our instructions:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Op Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>000</td>
<td>ACC := ACC + (A)</td>
</tr>
<tr>
<td>SUB</td>
<td>001</td>
<td>ACC := ACC - (A)</td>
</tr>
<tr>
<td>LOAD</td>
<td>010</td>
<td>ACC := (A)</td>
</tr>
<tr>
<td>STORE</td>
<td>011</td>
<td>(A) := ACC</td>
</tr>
<tr>
<td>JUMP</td>
<td>100</td>
<td>PC := A</td>
</tr>
<tr>
<td>JZER</td>
<td>101</td>
<td>IF ACC==0 JUMP A</td>
</tr>
</tbody>
</table>

- A – our operand address (lower 13 bits of the instruction)

Instruction Decoding

- The opcode needs to be decoded so we can determine what portions (“subroutines”) of the microprogram apply.
- One method: a decoding tree
  - microprogram makes as many comparisons as there are opcode bits:
Instruction Decoding, cont.

• The decoding tree is very time consuming.
• In reality, other methods are used:
  – jump tables
    • jump to the table
    • jump to the spot in the table with the address
    • jump to the address
  – mapping memory
    • opcode goes to special memory to find the start address.

The Microprogram

• in web handout

MicroInstruction Format Types

• Horizontal microinstructions – there are no coded fields. There is one bit in the microinstruction for each signal.
• Vertical microinstructions – contain coded fields (example: using a 3-bit address field to specify which one of 8 registers should be used).
• Vertical microinstructions save on control memory but are slower (need to decode the fields).
• Most microarchitectures (such as ours) use a mixture of formats – some fields are coded, others are not.

Our Microinstructions

• Our control store is a 64x27 bit read-only memory.
• Microinstruction format (and #bits/field) is:
  • picture from handout
MicroInstruction Fields

• handout

Microprogram Syntax

• To be able to read our microprogram, we use a micro assembly language (MAL)
• One microinstruction per line, with different parts of the microinstruction separated by semicolons
• They are ordered on a line in the order in which they are carried out.
• Assigning a value, use “:=”
  \[ \text{MAR} := \text{PC} \]
• Jumps within the microcode are indicated by goto <line#>
  – goto 25

Microprogram Syntax, cont.

• Conditionals are written using if <condition> then
  – if N then goto 25 ; if negative go to 25
• ALU functions:
  – addition
    \[ \text{ACC} := \text{ACC} + 1 \]
  – and
    \[ \text{PC} := \text{and} (\text{IR}, \text{AMASK}) \]
  – pass (default)
    \[ \text{TMP} := \text{TMP} \]
  – complement (1’s complement)
    \[ \text{TMP} := \text{com}(\text{MBR}) \]
• Shifting:
  – right shift
    \[ \text{TMP} := \text{rshift}(\text{IR}) \]
  – left shift
    \[ \text{TMP} := \text{lshift}(\text{IR}) \]

Microprogram Syntax, cont.

• ALU and Shift functions can be combined:
  – adding, then shifting:
    \[ \text{TMP} := \text{lshift}(\text{IR} + \text{IR}) \]
  – adds IR and IR, then shifts left and stores the result in TMP
• An ALU function and a conditional jump can be combined on one line:
  \[ \text{TMP} := \text{TMP}; \text{if} \ N \text{ then goto 21} \]