## Lecture 24: Sample MicroArchitecture

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

## About the Sample MicroArchitecture

- Sample Microarchitecture – a learning tool
- Generated from multiple sources (by Prof. Hamel)
- Why not use Intel?

## Control Unit

- Control unit –
- MAP –
- Control memory (Control store) –
- MUX conditional codes –

## ALU

- ALU has three components:
- ALU Functions:

## ALU, cont.

- Shift functions:

- 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):
  - PC (Register 1):
  - IR (Register 2):
  - TMP (Register 3):
  - AMASK (Register 4):
  - 1 (Register 5):

## MicroArchitecture Registers

- Other Registers:
  - MAR:
  - MBR:
  - MIR:
  - MPC:

## 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.

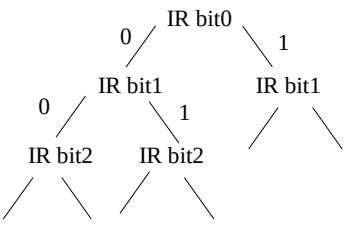
- Our instructions:

| Instruction | Op Code | Description |
|---|---|---|
| ADD | 000 | ACC := ACC + (A) |
| SUB | 001 | ACC := ACC – (A) |
| LOAD | 010 | ACC := (A) |
| STORE | 011 | (A) := ACC |
| JUMP | 100 | PC := A |
| JZER | 101 | IF ACC==0 JUMP A |

- 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.

- Time consuming!
- More common methods:
  - 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 –
- Vertical microinstructions –
- Mixed

## 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

- Micro Assembly Language (MAL)
- One microinstruction per line
- Parts of instruction are separated by semi-colons and given in the order in which they are carried out.
- Assigning a value, use ":="
  MAR := PC
- Jumps within the microcode are indicated by goto <line#>
  - goto 25

## Microprogram Syntax, cont.

- Conditionals are written using if <condition> then
- ALU functions:
  - addition
  - and
  - pass (default)
  - complement (1's complement)

## Microprogram Syntax, cont.

- Shifting:
  - right shift
  - left shift

## Microprogram Syntax, cont.

- ALU and Shift functions can be combined:
  - adding, then shifting:
    TMP := lshift(IR + 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:
  TMP := TMP; if N then goto 21