Levels of Programming

- High level - C, C++, Java et al
  furthest removed from HW
  compiler translates high level constructs into machine code
  translation may or may not be optimal in time
  sometimes use assembly language to facilitate this optimization

- Low level - assembly language
  closer to HW; manipulate registers and memory directly
  often fastest executable since can choose other ways to manipulate data
  inefficient use of programming time

- Lowest level – microprogramming
  direct control over HW down to the bit level
  similar to assembly language in that can have loops, subroutines, branches
  typically have a smaller instruction set

Characterized By

- highly ordered systematic structure

- separate control signals from data

- architecture consists of control store (memory) which holds the microinstructions

- alter controller operation by modifying control store contents

- like a stored program – reminds one of ROM-based controller

Distinguishing Features Between Microprogramming and Assembly Language Programming

- level of control

- type of instruction set - µprogramming is like RISC

- parallelism is inherent in µprogramming; need parallel algorithms for assembly language

- generally much faster in that operate at HW speeds
ROM-based controller

System Architecture

Inputs

ROM

Next State

Outputs

Contents of ROM

( microinstruction format )

CK

Next State

Present State

R

E

G

Contents of ROM

( microinstruction format )

Inputs

Outputs

Next State

Present State

ROM specifications

Note that adding a single input doubles the number of ROM locations, since the number of locations = 2 exp (N_I + N_SV)

ROM inputs (address space) = N_I + N_SV

Where

N_I = number of system inputs

N_SV = number of state variables

N_O = number of system outputs

Types of microprogrammed implementations

• Multiple qualifiers (inputs) per state

  this is just the ROM-based controller shown above

  major problem is that the system inputs appear in the address space

• Single qualifier per state

  test only one input at a time – this changes the ASM chart and has timing ramifications

  a) dual address

  b) single address

For each implementation we must specify:

  a) system architecture

  b) microinstruction format

Summary of multiple qualifier per state implementation

  a) implement any ASM chart of arbitrary complexity

  b) results in much wasted memory as many input combinations never appear

  c) many microinstructions, but each is “narrow”

Single qualifier per state implementation

  a) test only one input at a time

  b) thus we must insert additional states between multiple tests

  c) no conditional outputs (replace them with states)