CPU design: Lecture 3

- The control unit
- This sequences and controls all the data movement and manipulation that implements the instruction set.

Copyright 1998-2004 © Leslie S. Smith

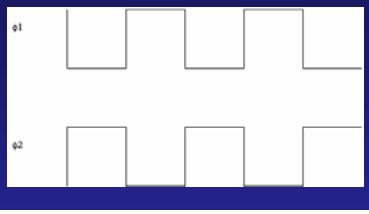
Generating the Control Unit

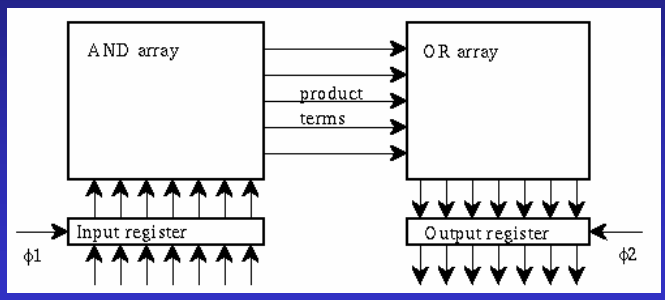
- We could use random logic to generate the control signals
  - many inputs and outputs,
    - hard to design, and difficult to ensure correctness
  - lacks any structure
  - difficult to produce an correct and efficient VLSI layout automatically
- or we could use a canonical representation of the logic
  - e.g. a PLA (programmable logic array)
  - including timing on as an input
    - since we need to generate a **sequence** of control signals
- What is a PLA (mentioned here: will return to this after CES lectures)
  - a way of implementing a logical function as a **sum of products**
    - disjunctive normal form
  - For example, Karnaugh maps give us functions in sum of products form
  - ...and PLAs can be used to implement this directly.

Copyright 1998-2004 © Leslie S. Smith

Using a PLA

- PLA has input clocked in at $\phi 1$, and output clocked out at $\phi 2$
  - signals $\phi 1$ and $\phi 2$ are non-overlapping antiphase clocks
- PLAs are
  - regular in structure
  - easy to simulate
  - easy to generate
  - minimisable





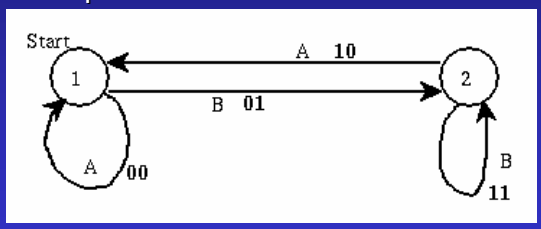
PLAs can be turned into

finite state machines

Copyright 1998-2004 © Leslie S. Smith

Finite State Machines

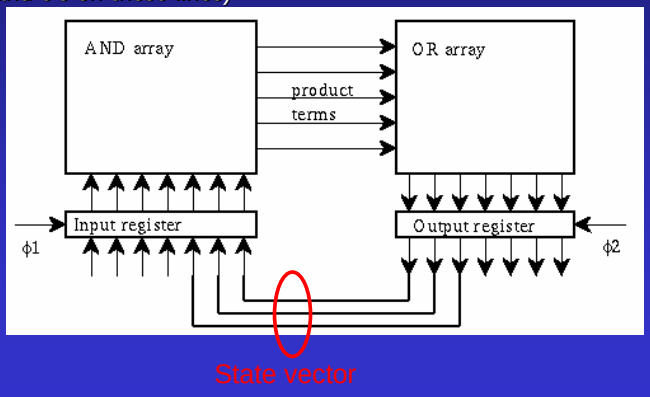
- A finite state machine (FSM) or finite state automaton (FSA) is a structure
  - with a finite number of states
  - which receives input **tokens**
  - and performs output **actions**
- They appear in many places in computer science
  - language design
  - parsing
  - protocol specification
  - even software design
- Example
  - Roman letters are input tokens
  - Bold numbers are output actions
  - One node is labelled start
- FSMs can be generated from PLAs
  - and used to implement the control unit



Copyright 1998-2004 © Leslie S. Smith

FSMs from PLAs

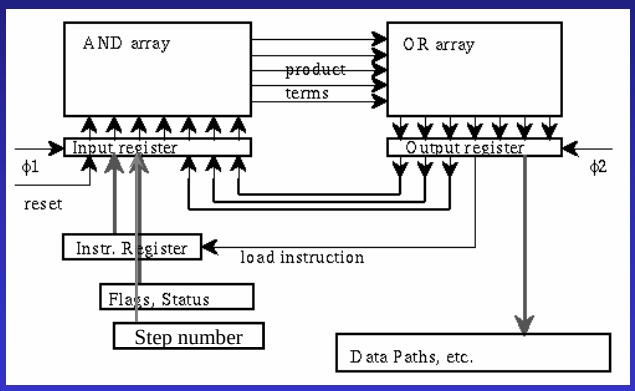
- We can make a PLA implement an FSA by feeding back some of the signals from the output back to the input.
- The system then has a **state**
  - the state is the state of the lines fed back
  - (I.e. the pattern of 1's and 0's on these lines)
- This FSA has
  - 3 feedback lines
  - 8 states (2^3 = 8)
- It can provide new control signals each clock tick



Copyright 1998-2004 © Leslie S. Smith

Controlling a CPU with an FSA

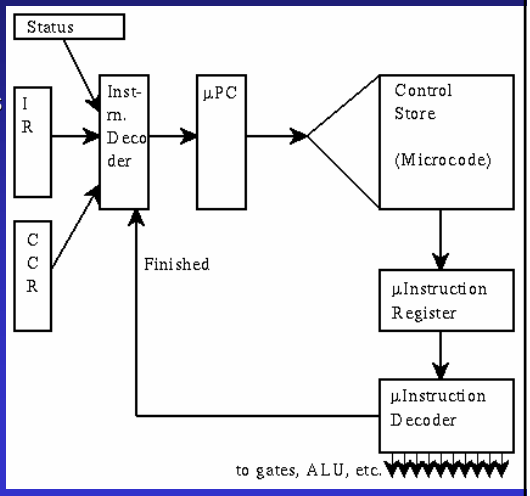
- Real FSAs are complex - implying complex PLAs
  - but there are tools for automating PLA design and for minimising PLAs.
- Hardwired control
  - is efficient
  - is fast
  - but may be expensive in silicon when CPU is complex.
  - This is becoming less crucial as feature-sizes diminish



Copyright 1998-2004 © Leslie S. Smith

Microprogramming

- Instructions are defined as a sequence of microinstructions.
  - Each microinstruction bit defines the opening and closing (etc) of a gate
  - (that is, of a gate in the datapath)
  - microinstruction program is a sequence of microinstructions.
  - IR decoder generates a start address
  - This instructions is placed in the Microprogram register
  - the Microprogram is executed sequentially until the finished output is asserted
  - A new instruction is fetched



Copyright 1998-2004 © Leslie S. Smith

Microcode and Microinstructions

- The microinstruction decoder is like the (normal) instruction decoder
  - it generates outputs
    - but these are much simpler than those from a full instruction decoder
  - finished is simply one of these outputs
- In its simplest and original form, each microinstruction bit corresponds to one control signal
  - proposed by Wilkes in 1951
  - but microcode is not always used in this way
- Microprogram memory needs to be very fast
  - in fact microprogrammed CPUs tend to be slower than hardwired controlled CPUs
  - but the control logic is much easier to design
    - particularly for CISC processors.
- The CPU instruction set is defined by the sequence of microcode instructions
  - one could (in theory) develop a new instruction set by writing new microcode

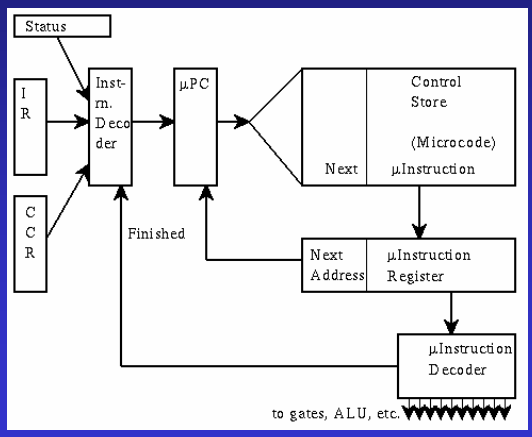
Copyright 1998-2004 © Leslie S. Smith

Microprogramming developments

- Control store is expensive
  - either in on-chip space
  - or because very fast SRAM is requires
- It is more efficient to share it between instructions.
- One way of doing this:
- have a *next* field in each microinstruction

This field identifies which microinstruction will be executed next

This allows some degree of sharing of microcode.



Copyright 1998-2004 © Leslie S. Smith

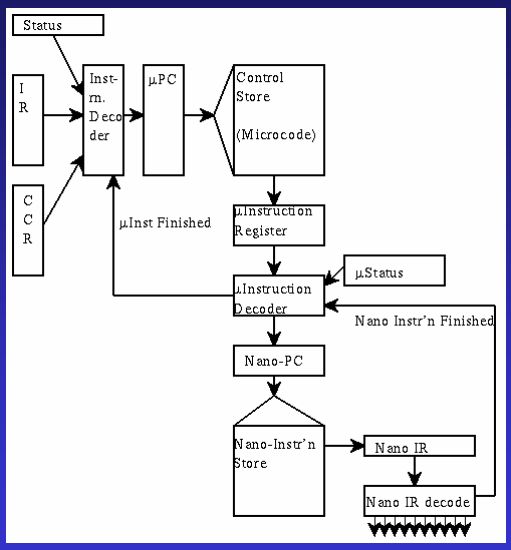
Sharing Microcode I

- More effective sharing of microcode is possible using a subroutining mechanism
  - a microinstruction stack is used
  - the microinstruction PC is stacked
  - the micro-instruction subroutine is called
  - the microinstruction PC is un-stacked
  - this is space-efficient because many instructions share identical elements
    - such as EA generation, etc.
  - but this is not time-efficient since each micro-routine call and return imposes a time overhead
    - and this may not always be acceptable

Copyright 1998-2004 © Leslie S. Smith

Sharing Microcode II

- An alternative technique is to
- use 2-level microcode
- The microinstruction register specifies the start of a sequence of lower-level microcode
  - usually called nano-code
- This is more efficient
  - no time is wasted in stacking etc.



Copyright 1998-2004 © Leslie S. Smith

Grouping Control Signals

- Using 1 bit per gate results in very long words of microcode
  - it makes the instructions very wide
  - but many possible combinations (= possible words of microcode) are never used, or are impossible
  - for example
    - one never has both read and write asserted on any register<-> bus link
    - one never writes more than one register to a bus a a time
    - one never reads the bus to more than one register at a time
- so sometimes the signals are grouped
  - that is, coded more efficiently, resulting in shorter microcode words
  - for example
    - connecting 31 registers bi-directionally to a single bus needs 62 control signals
      - implies 62 bits in a control word
    - but only one can be read, and one can be written at any one time.
      - So one can use $[\log_2 31]$ for each direction
      - that is, 10 bits
    - Why did we use only 31 registers, and not 32?

Copyright 1998-2004 © Leslie S. Smith

But...the Penalties associated with Grouping

- With ungrouped signals, no further decoding is required
  - the signals are applied directly to the gates
  - this is fast
- With grouped signals, the coded bit pattern needs to be decoded
  - and transformed into signals for the gates
  - this takes time
  - exactly how much time depends on quite how tightly the bit patterns were coded
- Using ungrouped microcode is known as **horizontal** microprogramming
- Using (tightly) grouped microcode is known as **vertical** microprogramming
- The same problem applies to ALU operations
  - using a tight coding for them saves space in the instruction
    - that means programs are shorter
    - and that reduces the CPU/Memory bottleneck...
  - but implies time is required for decoding them before the ALU operation can start

Copyright 1998-2004 © Leslie S. Smith