ESE534: Computer Organization

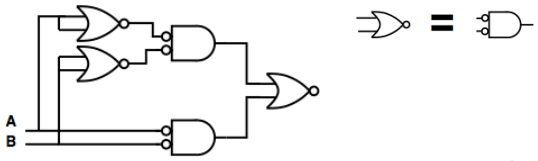
Day 6: February 3, 2010
Virtualization,
Programmable Architectures

Preclass Parity

- How many gates?
- Draw solutions

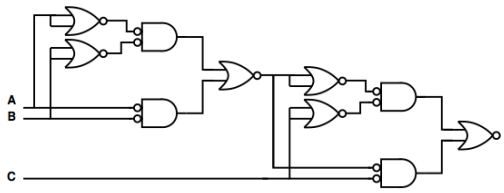
Preclass xor2 in nor2s

- Version shown on board not quite right.
- xor2(a,b) = /xnor2(a,b) = /(/a*/b+a*b)



Preclass xor3 in nor2s

- Xor3(a,b,c)=xor2(xor2(a,b),c)



Last Time

- Memory
- Memories pack state compactly
  - densely

Day 5

What is Importance of Memory?

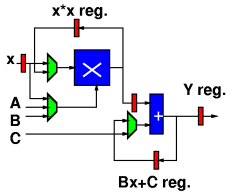
- Radical Hypothesis:
  - Memory is simply a very efficient organization which allows us to store data compactly
    - (at least, in the technologies we've seen to date)
  - A great engineering trick to optimize resources
- Alternative:
  - memory is a **primary**

Today

- Virtualization
- Programmable Gates
  - Muxes, ALUs
- Datapath Operation
- Memory
  - …continue unpacking the role of memory…

Last Wednesday

- Given a task: $y=Ax^2+Bx+C$
- Saw how to share primitive operators
- Got down to one of each



Very naively

- Might seem we need one of each different type of operator

..But

- Doesn't fool us
- We already know that **nand** gate
  (and many other things—HW1.B)
  …. are universal
- So, we know, we can build a universal compute operator

Temporal Composition

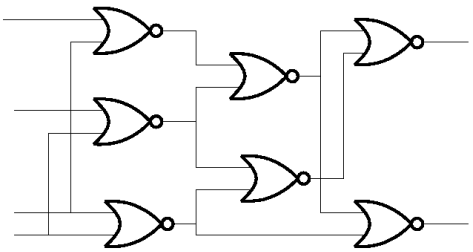
Temporal

- Don't have to implement all the gates *at once*
- Can *reuse* one gate over time

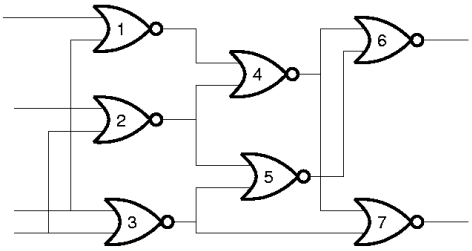
Temporal Decomposition

- Take Set of gates
- Sort topologically
  - All predecessors before successors
- Give a unique number to each gate
  - Hold value of its outputs
- Use a memory to hold the gate values
- Sequence through gates

Example Logic



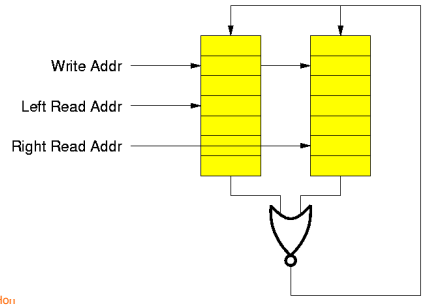
Numbered Gates



Preclass

- Number gates

nor2 Memory/Datapath



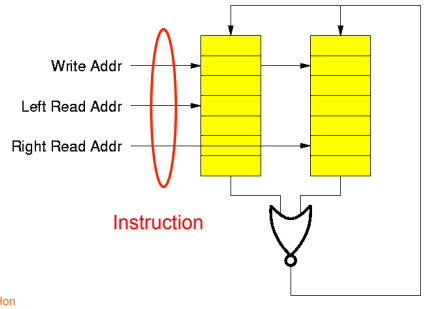
Programming?

- How do we program this network?

Programming?

- Program gates
  - Tell each gate where to get its input
    - Tell gate n where its two inputs come from
    - Specify the memory location for the output of the associated gate
  - Each gate operation specified with
    - two addresses (the input sources for gate)
    - This is the *instruction* for the gate

nor2 Memory/Datapath

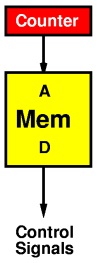


Supply Instruction

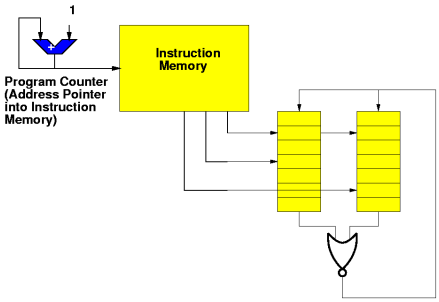
- How can we supply the sequence of instructions to program this operation?

Simplest Programmable Control

- Use a memory to "record" control instructions
- "Play" control with sequence



Temporal Gate Architecture



How program preclass computation?

Simulate the Logic

- For Preclass
- Go around the room calling out:
  - Identify PC
  - Identify instruction
    - Perform nor2 on slot __ and slot ___
  - Result is ___
  - Store into slot ___

What does this mean?

- With only one active component
  - **nor** gate
- Can implement **any** function
  - given appropriate
    - state (memory)
    - muxes (interconnect)
    - Control

Defining Terms

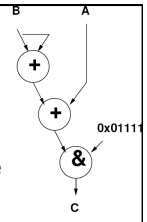
Fixed Function:

- Computes one function (e.g. FP-multiply, divider, DCT)
- Function defined at fabrication time

Programmable:

- Computes "any" computable function (*e.g.* Processor, DSPs, FPGAs)
- Function defined after fabrication

Result



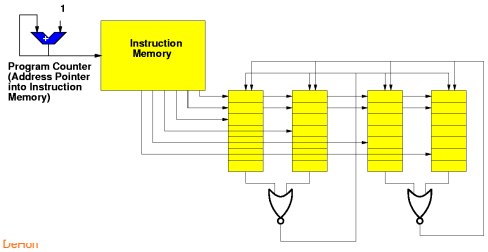
- Can sequence together primitive operations in time
- **Communicating** state through memory
  - Memory as interconnect
- To perform "arbitrary" operations

"Any" Computation? (Universality)

- Any computation which can "fit" on the programmable substrate
- **Limitations**: hold entire computation and intermediate data

Temporal-Spatial Variation

- Can have any number of gates
  - Tradeoff Area for Reduce Time….



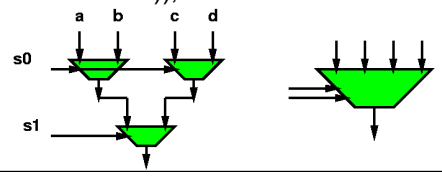
Use of Memory?

- What did we use memory for here?
- State
- Instructions
- Interconnect

Programmable Functions

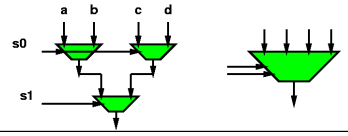
Mux can be a programmable gate

- bool mux4(bool a, b, c, d, s0, s1) {
  return(mux2(mux2(a,b,s0),
  mux2(c,d,s0),
  s1));
  }



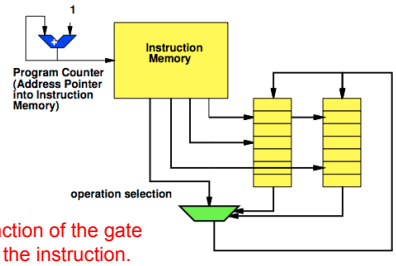
Mux as Logic

- bool and2(bool x, y)
  {return (mux4(false,false,false,true,x,y));}
- bool or2(bool x, y)
  {return (mux4(false,true,true,true,x,y));}
- Just by routing "data" into this mux4,
  - Can select **any** two input function



Programmable Variation

- Can use programmable gate in place of **nor** gate

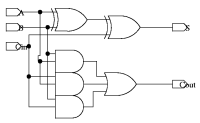


Specifying the function of the gate becomes part of the instruction.

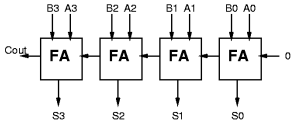
Is an Adder Universal?

- Assuming interconnect:
  - (big assumption as we'll see later)
  - Consider:

A: 001a
B: 000b
S: 00cd



- What's c?

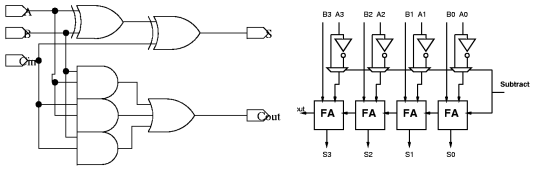


Practically

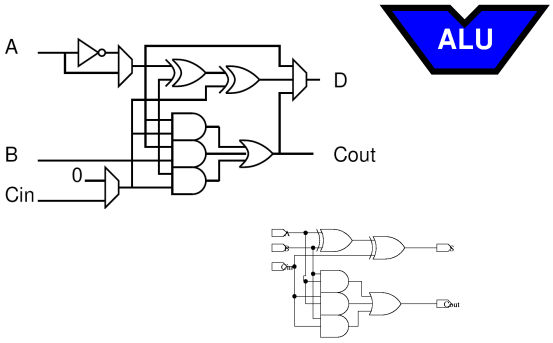
- To reduce (some) interconnect, and to reduce number of operations, do tend to build a bit more general "universal" computing function

Arithmetic Logic Unit (ALU)

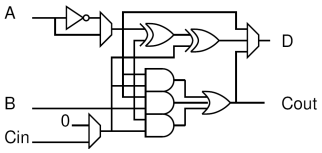
- Observe:
  - with small tweaks can get many functions with basic adder components



Arithmetic and Logic Unit

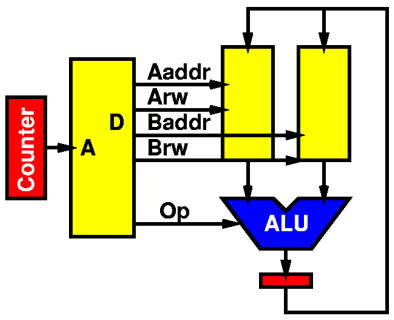


ALU Functions



- A+B w/ Carry
- B-A
- A xor B (squash carry)
- A*B (squash carry)
- /A

Slightly more conventional Programmable Architecture

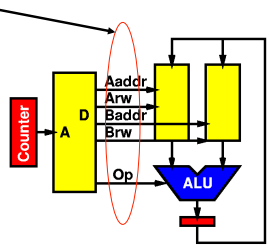


"Stored Program" Computer/ Processor

- Can build a datapath that can be *programmed* to perform **any** computation.
- Can be built with limited hardware that is *reused* in time.
- **Historically:** this was a key contribution from Penn's Moore School
  - Computer Engineers: Eckert and Mauchly
  - ENIAC→EDVAC
  - (often credited to Von Neumann)

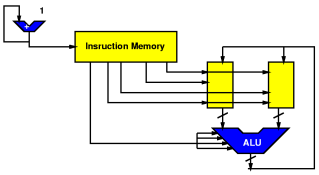
Instructions

- Identify the bits which control the function of our programmable device as:
  - *Instructions*



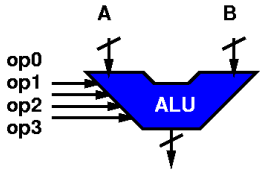
Programming an Operation

- Consider:
  - C = (A+2B) & 00001111
- Cannot do this all at once
  - But can do it in pieces…like nor2 case



ALU Encoding

- Each operation has some bit sequence
- ADD 0000
- SUB 0010
- INV 0001
- SLL 1110
- SLR 1100
- AND 1000

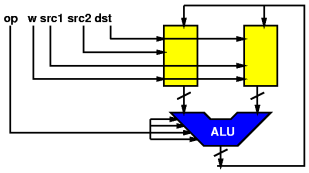


Programming an Operation

C = (A+2B) & 00001111

Op w src1 src2 dst

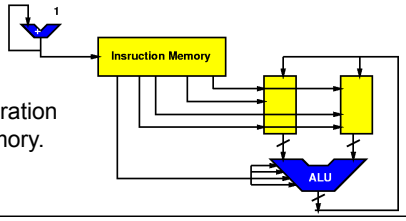
- Decompose into pieces
  - Compute 2B 0000 1 001 001 010
  - Add A and 2B 0000 1 000 010 011
  - AND sum with mask 1000 1 011 100 111



Fill Instruction Memory

Op w src1 src2 dst

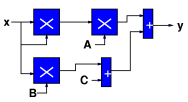
- 000: 0000 1 001 001 010
- 001: 0000 1 000 010 011
- 010: 1000 1 011 100 111



Program operation by filling memory.

What have we done?

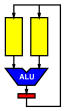
- Taken a computation: $y=Ax^2+Bx+C$
- Turned it into operators and interconnect



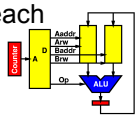
- Decomposed operators into a basic primitive:
  - nor, 2-input mux gates, adds, ALU

What have we done?

- Said we can implement it on as few as one of **compute unit** {ALU, mux, nor}
- Added a unit for **state**



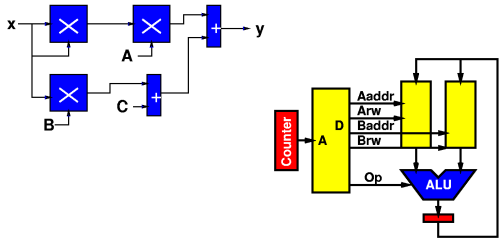
- Added an **instruction** to tell single, universal unit how to act as each operator in original graph



Virtualization

- We've *virtualized* the computation
- No longer need one **physical** compute unit for each operator in original computation
- Can suffice with:
  1. shared operator(s)
  2. a **description** of how each operator behaved
  3. a place to store the intermediate data between operators

Virtualization



Why Interesting?

- Memory compactness
- This works and was interesting because
  - the area to describe a computation, its interconnect, and its state
  - is much smaller than the physical area to spatially implement the computation
- *e.g.* traded multiplier for
  - few memory slots to hold state
  - few memory slots to describe operation
  - time on a shared unit (ALU)

Questions?

Admin Comments

- Day5 posted slides – section at end connecting preclass equations to memory model
- Office Hours Wednesday (today 2pm)
  - Comes
    - after homework goes out (Monday)
    - after lecture on material (Wednesday)
    - before homework is due (next Monday)

Big Ideas [MSB Ideas]

- Memory: efficient way to hold state
  - …and allows us to describe/implement computations of unbounded size
- State can be << computation [area]
- Resource sharing: key trick to reduce area
- Memory key tool for Area-Time tradeoffs
- "configuration" signals allow us to *generalize* the utility of a computational operator

Big Ideas [MSB-1 Ideas]

- ALUs and Muxes as universal compute elements
- First programmable computing unit
- Two key functions of memory
  - retiming (interconnect in time)
  - instructions
    - description of computation