ESE680-002 (ESE534): Computer Organization

Day 5: January 24, 2007
ALUs, Virtualization…

Last Time

• Memory
• Memories pack state compactly
  – densely

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

• ALUs
• Virtualization
• 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) are universal
- So, we know, we can build a universal compute operator

This Example

- \( y = Ax^2 + Bx + C \)
- Know a single adder will do

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)
- \( /A \)
- \( B < < 1 \)

Table Lookup Function

- Observe 2: only \( 2^3 = 256 \) functions of 3 inputs
  - 3-inputs = \( A, B, \) carry in from lower
- Two, 3-input Lookup Tables
  - give all functions of 2-inputs and a cascade
  - 8b to specify function of each lookup table

- LUT = LookUp Table

What does this mean?

- With only one active component
  - ALU, \( \text{nand} \) gate, LUT
- Can implement any function
  - given appropriate
    - state registers
    - 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

Revisit Example

Can implement any function given appropriate
- state registers
- muxes (interconnect)
- control

- We do see a proliferation of memory and muxes -- what do we do about that?

Virtualization
Back to Memories

- State in memory more compact than “live” registers
  - shared input/output/drivers
- If we’re sequentializing, only need one (few) data item at a time anyway
  - *i.e.* sharing compute unit, might as well share interconnect
- Shared interconnect also gives muxing function

What’s left?

Control

- Still need that controller which directed which state, went where, and when
- Has more work now,
  - also say what operations for compute unit

Implementing Control

- Implementing a single, fixed computation
  - might still just build a custom FSM

...and Programmable

- At this point, it’s a small leap to say maybe the controller can be programmable as well
- Then have a building block which can implement anything
  - within state and control programmability bounds
Simplest Programmable Control

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

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

ALU Encoding

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

Programming an Operation

\[
C = (A+2B) \& 00001111
\]

- 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.

Machine State: Initial

- Counter: 0
- Instruction Memory:
  - 000: 0000 1 001 001 010
  - 001: 0000 1 000 010 011
  - 010: 1000 1 011 100 111
- Data Memory:
  - 000: A
  - 001: B
  - 010: ?
  - 011: ?
  - 100: 00001111
  - 101: ?
  - 110: ?
  - 111: ?

First Operation

- Counter: 0
- Instruction Memory:
  - 000: 0000 1 001 001 010
  - 001: 0000 1 000 010 011
  - 010: 1000 1 011 100 111
- Data Memory:
  - 000: A
  - 001: B
  - 010: ?
  - 011: ?
  - 100: 00001111
  - 101: ?
  - 110: ?
  - 111: ?

First Operation Complete

- Counter: 0
- Instruction Memory:
  - 000: 0000 1 001 001 010
  - 001: 0000 1 000 010 011
  - 010: 1000 1 011 100 111
- Data Memory:
  - 000: A
  - 001: B
  - 010: 2B
  - 011: ?
  - 100: 00001111
  - 101: ?
  - 110: ?
  - 111: ?

Update Counter

- Counter: 1
- Instruction Memory:
  - 000: 0000 1 001 001 010
  - 001: 0000 1 000 010 011
  - 010: 1000 1 011 100 111
- Data Memory:
  - 000: A
  - 001: B
  - 010: 2B
  - 011: ?
  - 100: 00001111
  - 101: ?
  - 110: ?
  - 111: ?

Second Operation

- Counter: 1
- Instruction Memory:
  - 000: 0000 1 001 001 010
  - 001: 0000 1 000 010 011
  - 010: 1000 1 011 100 111
- Data Memory:
  - 000: A
  - 001: B
  - 010: 2B
  - 011: ?
  - 100: 00001111
  - 101: ?
  - 110: ?
  - 111: ?
Second Operation Complete

- Counter: 1
- Instruction Memory:
  000: 0000 1 001 001 010
  001: 0000 1 000 010 011
  010: 1000 1 011 100 111

- Data Memory:
  000: A
  001: B
  010: 2B
  011: A+2B
  100: 00001111
  101: ?
  110: ?
  111: ?

Update Counter

- Counter: 2
- Instruction Memory:
  000: 0000 1 001 001 010
  001: 0000 1 000 010 011
  010: 1000 1 011 100 111

- Data Memory:
  000: A
  001: B
  010: 2B
  011: A+2B
  100: 00001111
  101: ?
  110: ?
  111: ?

Third Operation

- Counter: 2
- Instruction Memory:
  000: 0000 1 001 001 010
  001: 0000 1 000 010 011
  010: 1000 1 011 100 111

- Data Memory:
  000: A
  001: B
  010: 2B
  011: A+2B
  100: 00001111
  101: ?
  110: ?
  111: ?

Third Operation Complete

- Counter: 2
- Instruction Memory:
  000: 0000 1 001 001 010
  001: 0000 1 000 010 011
  010: 1000 1 011 100 111

- Data Memory:
  000: A
  001: B
  010: 2B
  011: A+2B
  100: 00001111
  101: ?
  110: ?
  111: (A+2B) & ...

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
What have we done?

- Taken a computation: \[ y = Ax^2 + Bx + C \]
- Turned it into operators and interconnect
- Decomposed operators into a basic primitive: Additions, ALU, ...nand

What have we done?

- Said we can implement it on as few as one of compute unit \{ALU, LUT, nand\}
- 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

• Homework #2 due Monday
• Reading: Dennard on Scaling

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 LUTs as universal compute elements
• First programmable computing unit
• Two key functions of memory
  – retiming (interconnect in time)
  – instructions
    • description of computation