CS184a: Computer Architecture (Structures and Organization)

Day5: October 9, 2000

Computing Requirements;

Instruction Taxonomy

Last Time

- Generalize compute elements/datapaths for reuse
- Instructions to tell datapaths how to behave
- Memories pack state compactly
- Memories can serve as programmable control
- Virtualize computing operators
  - store descript. and state compactly in memory
  - generalize and reuse datapath

Today

- Memory
  - unbounded
  - impact on computability
- Computing Requirements (Review)
- Instruction Distribution Requirements
- Instruction Taxonomy

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

"Any" Computation? (Universality)

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

Motivating Questions

- What is required for recursion?
- What is the role of
  - new
  - malloc
  - cons

- Consider
  - routine to produce an n-element vector sum
  - downloading an image off the web
  - decompressing a downloaded file
  - read input string from user

"Any" Computation

- Computation can be of any size
- Consider UTM with unbounded input tape to describe computation

Computation Evolves During Execution

- Conventional think:
  - program graph unfolds with
    - procedure calls
    - thread spawns
  - unfold state with
    - new
    - malloc

Computing Evolves During Execution

- What's happening?
  - New, malloc -- allocating new state for virtual operators
  - procedure calls and spawns -- unfolding the actual compute graph
    - from a range of possible graphs
  - use computation to **define** the computation

Example

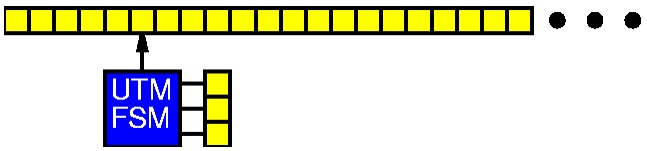
- Vsum(a,b)
  - c = new int[a.length()];
  - for(I=0;I<a.length();I++)
    - c[I]=a[I]+b[I];
  - return(c)
- Vsum4(a,b,c)
  - c[0]=a[0]+b[0];
  - c[1]=a[1]+b[1];
  - c[2]=a[2]+b[2];
  - c[3]=a[3]+b[3];

Memory Function

- Allow unbounded computation
- Allow computational graph to evolve during computation

Computational Strength

- With memory appropriately arranged:
  - can now compute unbounded computations
  - …but finite
- As close as we'll come to a Turing Machine



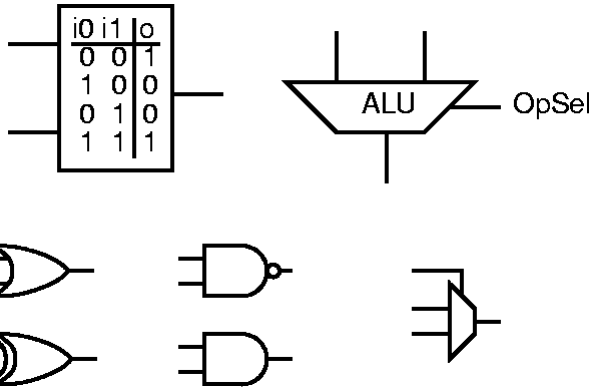
Computing Capability Review

- Gates:
  - boolean logic
  - finite functions
- Gates and registers:
  - Finite Automata
  - some infinite functions
- Memories with allocation
  - unbounded functions
  - TM w/in the limits of available memory

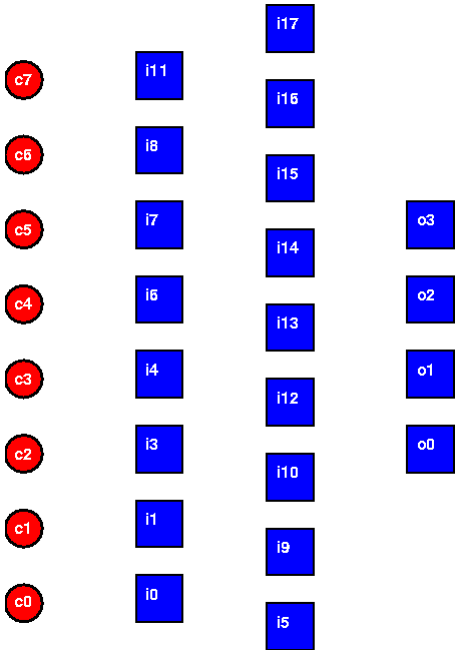
Computing Requirements (review)

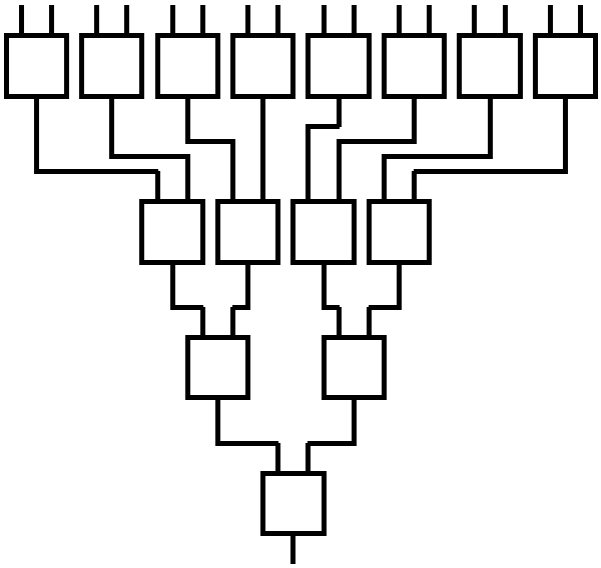
Requirements

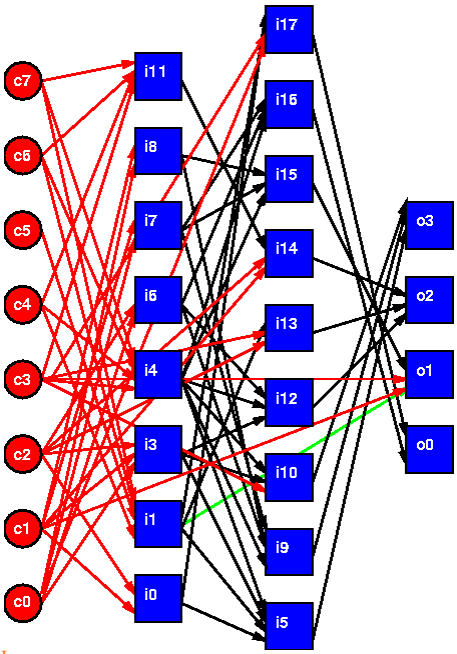
- In order to build a general-purpose computing device, we absolutely must have?
  - _
  - _
  - _
  - _
  - _



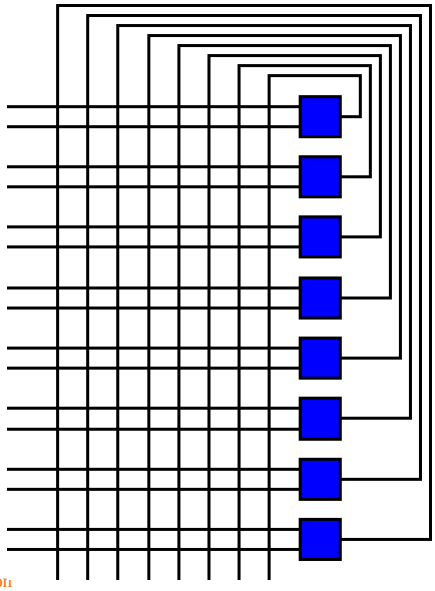
Primitive compute elements enough?



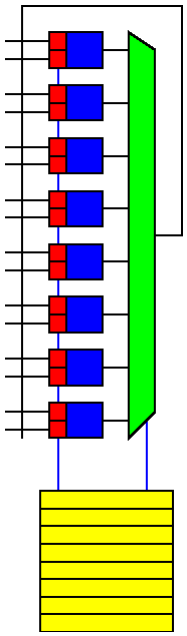



c7
c6
c5
c4
c3
c2
c1
c0
i11
i8
i7
i6
i4
i3
i1
i0
i17
i16
i15
i14
i13
i12
i10
i9
i5
o3
o2
o1
o0

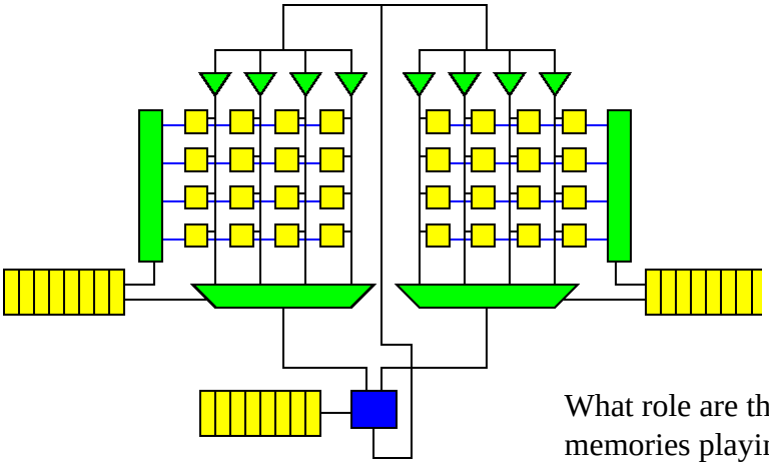
Compute and Interconnect



Sharing Interconnect Resources



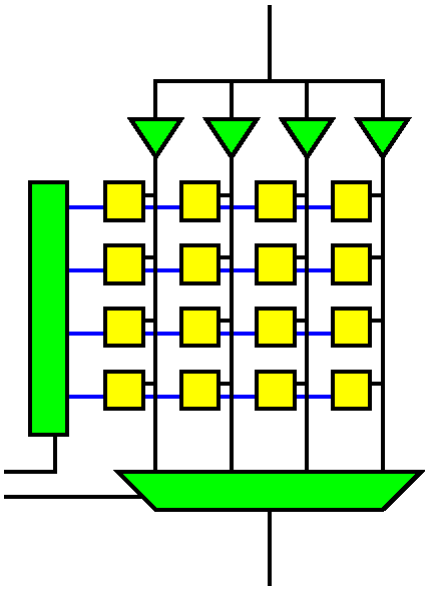
Sharing Interconnect and Compute Resources



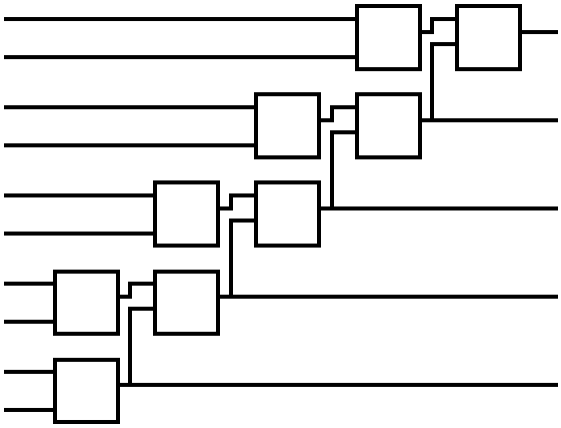
What role are the memories playing here?

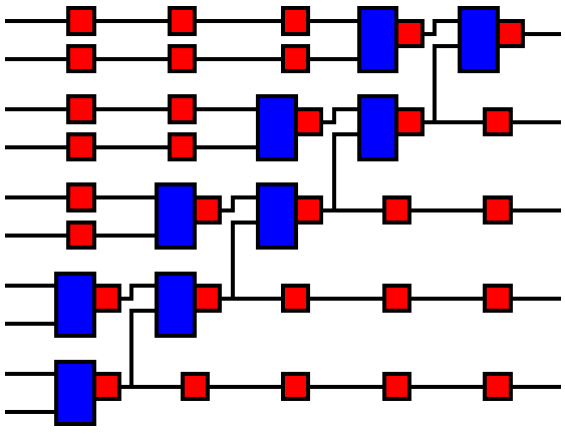
Memory block or Register File

Interconnect:
moves data from input to storage cell;
or from storage cell to output.



What do I need to be able to use this circuit properly?
(reuse it on different data?)





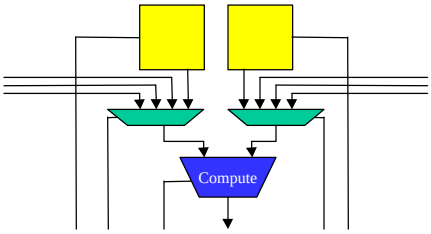
Requirements

- In order to build a general-purpose computing device, we absolutely must have?
  - Compute elements
  - Interconnect: space
  - Interconnect: time (retiming)
  - Interconnect: external (IO)
  - Instructions

Instruction Taxonomy

Instructions

- Distinguishing feature of programmable architectures?
  - *Instructions* -- bits which tell the device how to behave

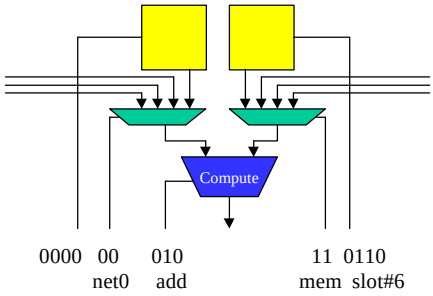


Focus on Instructions

- Instruction organization has a large effect on:
  - size or compactness of an architecture
  - realm of efficient utilization for an architecture

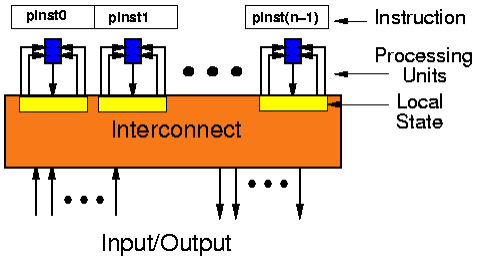
Terminology

- **Primitive Instruction (*pinst*)**
  - Collection of bits which tell a single bit-processing element what to do
  - Includes:
    - select compute operation
    - input sources in space
      - (interconnect)
    - input sources in time
      - (retiming)



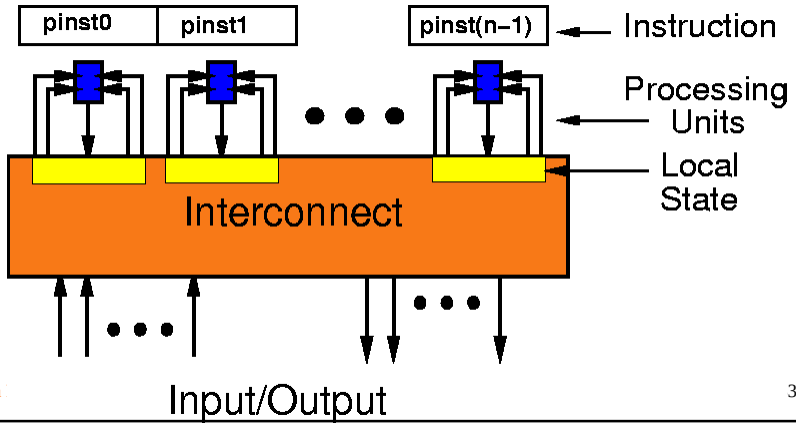
Computational Array Model

- Collection of computing elements
  - compute operator
  - local storage/retiming
- Interconnect
- Instruction



"Ideal" Instruction Control

- Issue a new instruction to every computational bit operator on every cycle



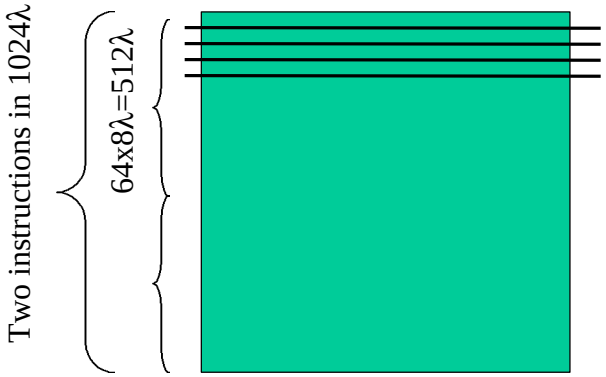
"Ideal" Instruction Distribution

- Why don't we do this?

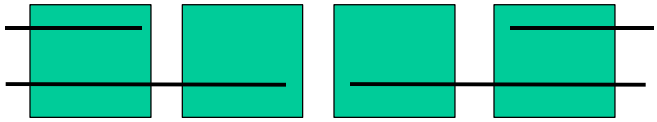
"Ideal" Instruction Distribution

- **Problem:** Instruction bandwidth (and storage area) quickly dominates everything else
  - Compute Block ~ $1M\lambda^2$ ($1K\lambda$ x $1K\lambda$)
  - Instruction ~ 64 bits
  - Wire Pitch ~ 8λ
  - Memory bit ~ $1.2K\lambda^2$

Instruction Distribution

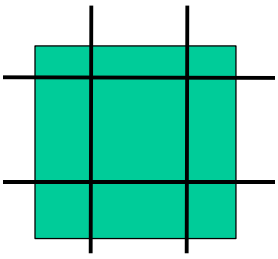


Instruction Distribution



Distribute from both sides = 2x

Instruction Distribution



Distribute X and Y = 2x

Instruction Distribution

- Room to distribute 2 instructions across PE per metal layer (1024 = 2×8×64)
- Feed top and bottom (left and right) = 2×
- Two complete metal layers = 2×

- ⇒ 8 instructions / PE Side

Instruction Distribution

- Maximum of 8 instructions per PE side
- Saturate wire channels at $8\times\sqrt{N} = N$
- ⇒ at 64 PE
  - beyond this:
    - instruction distribution dominates area

- Instruction consumption goes with area
- Instruction bandwidth goes with perimeter

Instruction Distribution

- Beyond 64 PE, instruction bandwidth dictates PE size

$$\frac{\sqrt{PE_{area}} \times 4 \times \sqrt{N}}{(64 \times 8\lambda)} = N$$

$$PE_{area} = 16K\lambda^2 \times N$$

- Build larger arrays

  $\Rightarrow$processing elements become less dense

Instruction Memory Requirements

- **Idea:** put instruction memory in array
- **Problem:** Instruction memory can quickly dominate area, too
  - Memory Area = $64 \times 1.2K\lambda^2$/instruction
  - $PE_{area} = 1M\lambda^2 + (\text{Instructions}) \times 80K\lambda^2$

Instruction Pragmatics

- Instruction requirements *could* dominate array size.
- Standard architecture trick:
  - Look for structure to exploit in "typical computations"

Typical Structure?

- What structure do we usually expect?

Two Extremes

- SIMD Array (microprocessors)
  - Instruction/cycle
  - share instruction across array of PE s
  - uniform operation in space
  - operation variance in time
- FPGA
  - Instruction/PE
  - assume temporal locality of instructions (same)
  - operation variance in space
  - uniform operations in time

Finishing Up...

Coming Attraction: Final Talk by Tom Knight

- Computing with Life
  - TODAY @ 3pm
  - Beckman Institute Auditorium
  - (biological computers)
- MPEGs of other two talks available
  - currently in /home/andre/tk*.mpg

Big Ideas [MSB Ideas]

- Basic elements of a programmable computation
  - Compute
  - Interconnect
    - (space and time, outside system [IO])
  - Instructions
- For unbounded computation
  - computational graph evolve/computed by computation

Big Ideas [MSB Ideas]

- Instruction resources can be significant
  - dominant/limiting resource
- Applications typically have structure
- Exploit this structure to reduce resource requirements
- Architecture is about understanding and exploiting structure and costs to reduce requirements

Big Ideas [MSB-1 Ideas]

- Two key functions of memory
  - retiming
  - instructions
    - description of computation