

Computer Systems Architecture II

Chip-Multiprocessors: Applications and Architectures

CSE 561M

Prof. Patrick Crowley

Plan for Today

- Announcements
 - Instead of meeting on April 22 & 24, we will meet on May 1 for project presentations
- Questions
- Review project logistics
- Paper presentations and discussions

Project Logistics

- Weekly Milestones
 - Milestone report due each week
 - Submit proposal Feb 26
 - Final presentation May 1
- Focus this week:
Reports and code freeze

Project Milestones

<i>M0</i>	<i>Feb 26</i>	<i>Project Proposal</i>
<i>M1</i>	<i>Mar 4</i>	<i>Design</i>
<i>M2</i>	<i>Mar 18</i>	<i>Implementation 1</i>
<i>M3</i>	<i>Mar 25</i>	<i>Implementation 2</i>
<i>M4</i>	<i>Apr 1</i>	<i>Implementation 3</i>
<i>M5</i>	<i>Apr 8</i>	<i>Implementation 4</i>
<i>M6</i>	<i>Apr 15</i>	<i>Wrap-up, Prepare reports</i>
<i>M7</i>	<i>Apr 24</i>	<i>Reports & code due via email</i>
<i>M8</i>	<i>May 1</i>	<i>Presentations/Last meeting</i>

Project Reports

- Contents
 - Problem statement, description & background
 - Your design
 - Your implementation
 - Include characteristics of your code, such as code size, ME utilization, memory utilization, etc.
 - Experimental results
 - Demonstrate correct operation
 - Measure and explain the performance of your code
 - Future work (that you might do if you had time)
 - Division of labor amongst group members
- Length: $5 \leq \# \text{ single-spaced pages} \leq 15$
- PDF is due via email on Thursday, April 24
 - Project code also due at that time

Project Presentations

- Project Presentation
 - 15 minutes
 - All group members should contribute
 - Content:
 - Subset of project report
 - What you did, why you did it, and how well it worked
 - Plan a live ONL demo
- May 1, during our final exam slot

Paper Presentations

- Each paper discussion will involve
 - A 15-minute **group** presentation
 - Summarize the paper's content and contributions
 - Relate it to your project (e.g., what if you had targeted that system for your project rather than the IXP?)
 - All group members should contribute
 - A 15 minute follow-up lecture
- Each group has an assigned paper & date
- Submit presentation via email by 2pm that day

Paper Discussions

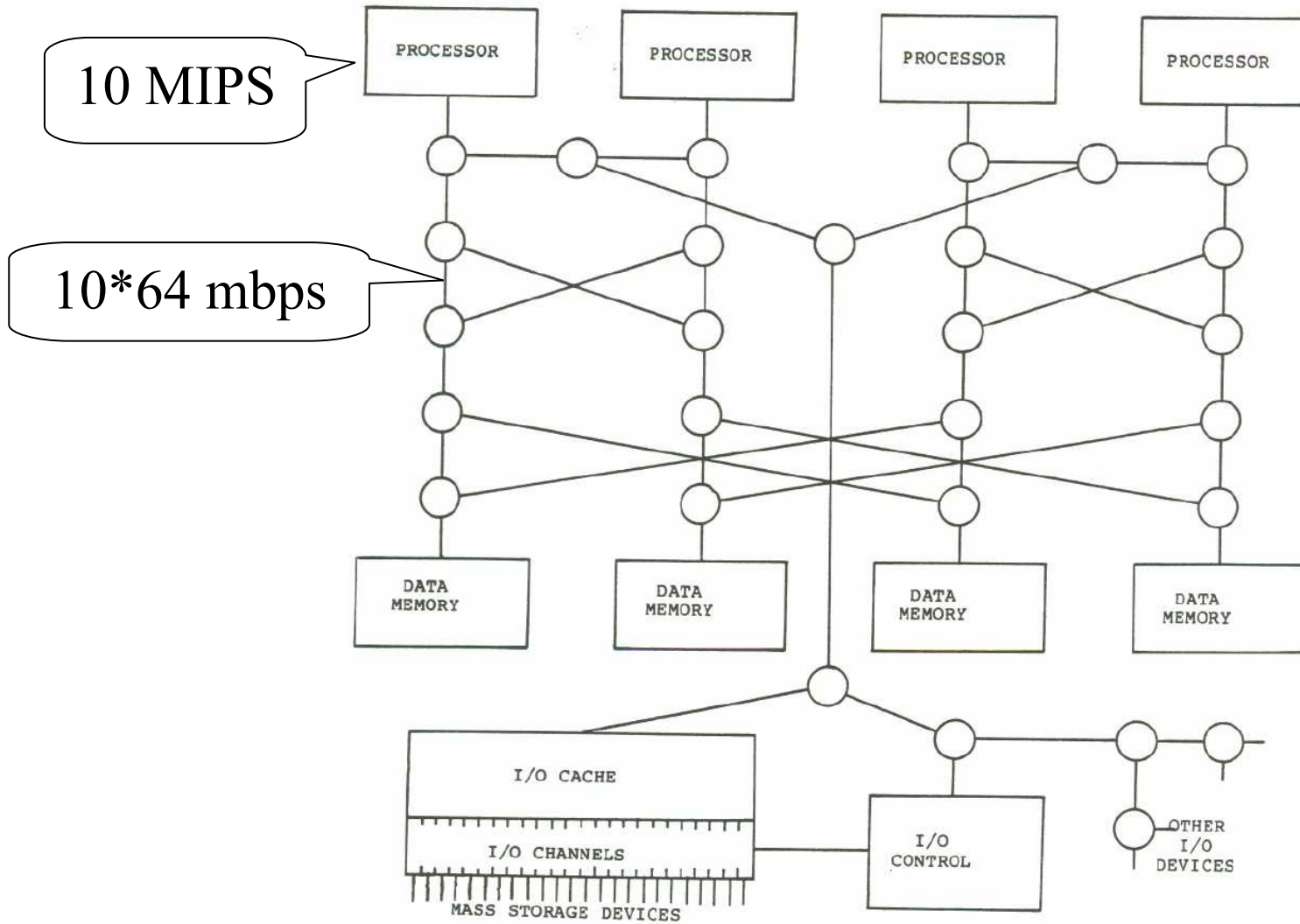
- Smith's Architecture and Applications of the HEP Multiprocessor Computer System
 - Presenters: Steve Nann, Tam Vu Ngoc, Dan Vianello
- Dennis and Misunas' A Preliminary Architecture for a Basic Data-Flow Processor
 - Presenters: Steve Prochazka, Mark Dunn
- Shriraman et al.'s Flexible Decoupled Transactional Memory Support
 - Presenters: Eitan Marder-Eppstein, Stu Glaser

*Architecture and Applications of
the HEP Multiprocessor
Computer System*

HEP

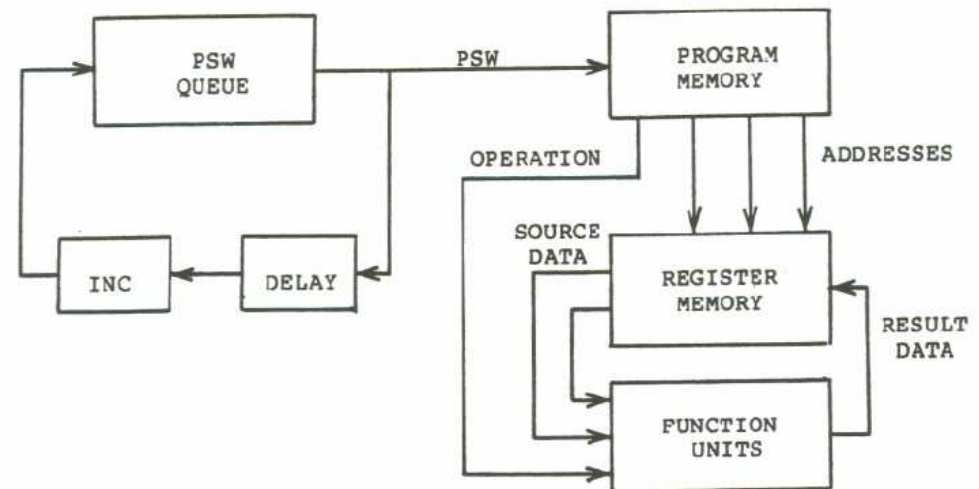
- Heterogeneous Element Processor
- Large, parallel computer built for scientific applications
- Processors are pipelined, each stage holds an instruction from a different thread
- System consists of processors, switches, links, memories and I/O units

System Organization



Processor Organization

- Process status word (PSW) contains 20-bit PC
- 2K, 64-bit general purpose registers
- There is a minimum 8 cycle latency between instructions from a given thread
 - Need 8 threads for full utilization



Loads and Stores

- The scheduler function unit (SFU) moves data between memory and registers
- On load/store, the SFU
 - Sends a packet with the address, return address for processor and process ID, and 64-bits of data (for a store)
 - Removes the requesting PSW from the control queue, adds it to its own stalled queue
 - Handles the response packet

Control Flow

- Conditional branches can be implemented by storing, loading and modifying the current PSW
- Instructions also exist to create and terminate threads (i.e., processes)

Managing Shared Data

- Each data memory location is either “empty” or “full”
- Each register is “empty”, “full” or “reserved”
- A location is emptied when it is read, and filled when it is written
 - A load, e.g., can wait for a location to be full before consuming the data

Processes, Tasks and Jobs

- A set of processes with the same protection domain is a *task*
- A *job* consists of one or more tasks
 - Tasks within a job have disjoint registers and program memory, but share data memory
- Each job must indicate its max number of active tasks, the OS only loads the job when sufficient resources are available

Interconnection Network

- Each switch has 3 bidirectional ports
- Each port has its own routing table, which maps dest addresses to output ports
 - these are populated during system configuration
- When two or three packets need to use the same output port, the highest priority packet is sent, and the others are routed elsewhere (no buffering)
- Packet priority increases with each re-direction
- Switch nodes check packet parity, errors are reported to a diagnostic subsystem

I/O

- User processes make I/O requests via supervisor calls
 - implement file systems, page mappings, etc.
- The high speed I/O systems consists of
 - *I/O cache modules*, which attach to the switch via up to 32
 - *I/O channels*, which carry data at 1.2MB/s, and are controlled by
 - *I/O control processors*, which accept requests from processes (via memory locations) and handles completed I/O requests from channels

Files

- When a file is opened
 - An explicit number of frames are allocated in the I/O cache
 - A sequential direction is indicated (for future reads/writes); most I/O instructions have their effect in this direction
 - A random I/O operation is available to modify the current file position explicitly
- The supervisor process manages cache contents explicitly
 - Determines the page containing the requested data, moves those pages into the cache
 - Schedules read-ahead, write-behind with I/O control processor
 - A reference count is held for each cached page; when count is zero, the page can be purged (needed when files are shared between processes)

HEP Programming

- Fortran specific
 - CREATE statement spins a subroutine call off into its own thread
 - Normal subroutines can become their own thread with the RESUME statement
 - A special syntax (\$ prefix) that allows access to full/empty variables for data sharing
 - Registers and local vars are allocated dynamically, so any routine can be called in any thread, at any time
- Synchronization can be achieved via empty/full states

Intellectual Digestion

- How does this flavor of multithreading compare to that of the MEs?
- Compare the HEP interconnection network to the interconnection approach used in the IXP. What are their relative strengths?
- Assuming similar I/O resources and learning curve, how would your project implementation have gone with the HEP? Faster/Slower? Harder/Easier? Better/Worse?

Multithreading

- HEP
 - Fine-grained (“context-switch” each cycle)
 - Dynamic number of threads, under program control
- IXP
 - Coarse-grained (voluntary context switches)
 - 8 contexts per ME

Interconnection

- IXP
 - On one chip, links are metal layers
 - Uses several buses
 - Good match for VLSI
- HEP
 - Links are cables
 - Switched network
 - Field-extendable

Your Project

- Assuming similar I/O resources and learning curve, how would your project implementation have gone with the HEP?
 - Faster/Slower?
 - Harder/Easier?
 - Better/Worse?

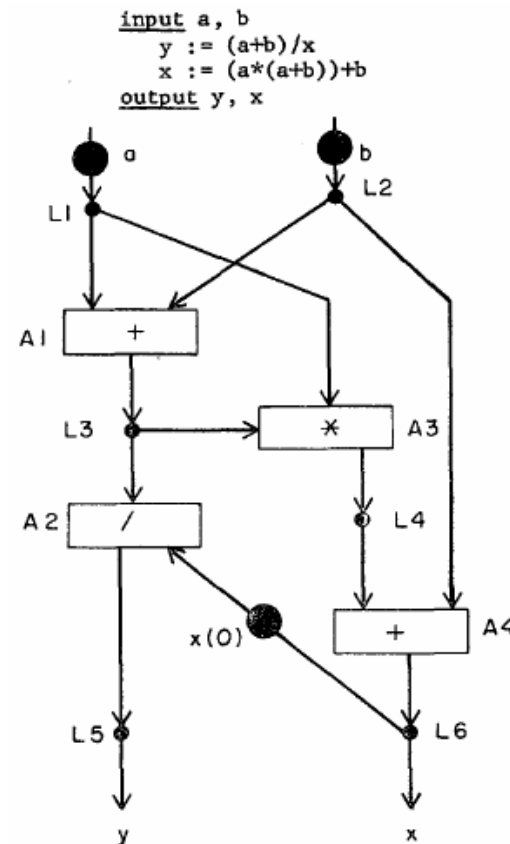
*A Preliminary Architecture for
a Basic Data-Flow Processor*

Data Flow

- The idea
 - Achieve concurrent program execution with a processor organization that allows an instruction to execute as soon as its operands are available
 - In contrast to the implied serial execution of traditional computers
- Most data flow proponents used new programming languages devoid of serial baggage
- A formal modeling technique, based on petri nets, inspired these ideas and can be used to derive formal properties of such systems

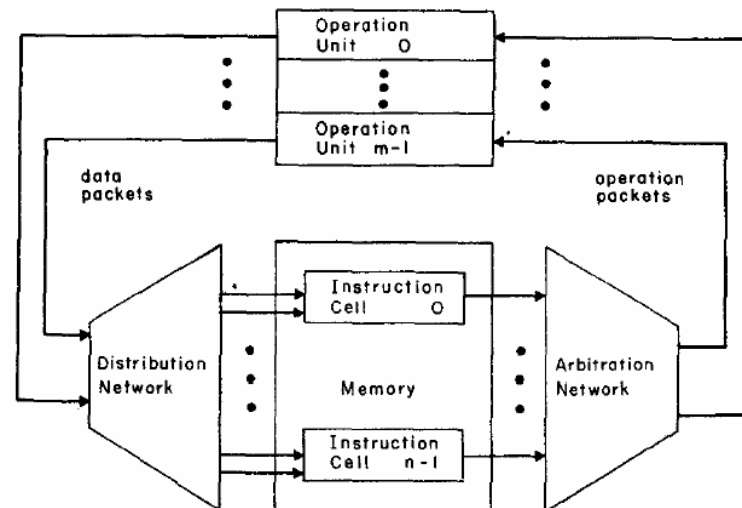
Sample Data Flow Program

- Nodes are operators or links (large dots = init vals)
- Arcs transmit values, represented by *tokens*
- A node is enabled only when all its inputs contain tokens
- A node may *fire* when it is enabled; tokens are removed from inputs, and one is placed on the output arc
- *Note: no control flow*



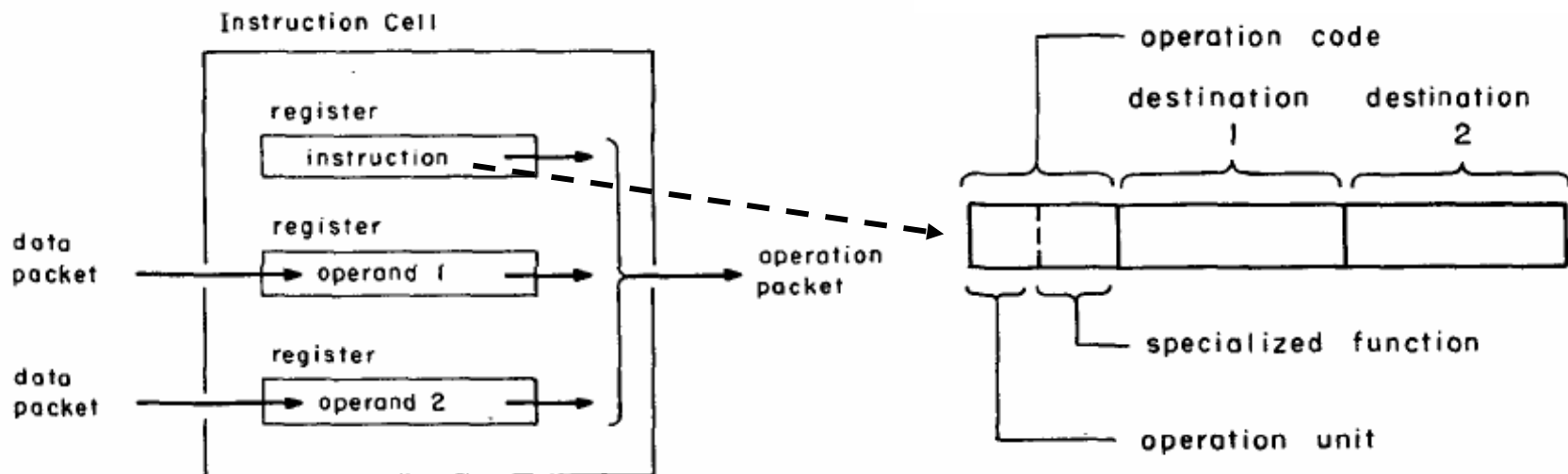
Elementary Processor

- Data flow program stored in *instruction cells*
- *Arbitration network* routes instruction cells to operation units
- *Distribution network* routes results to memory

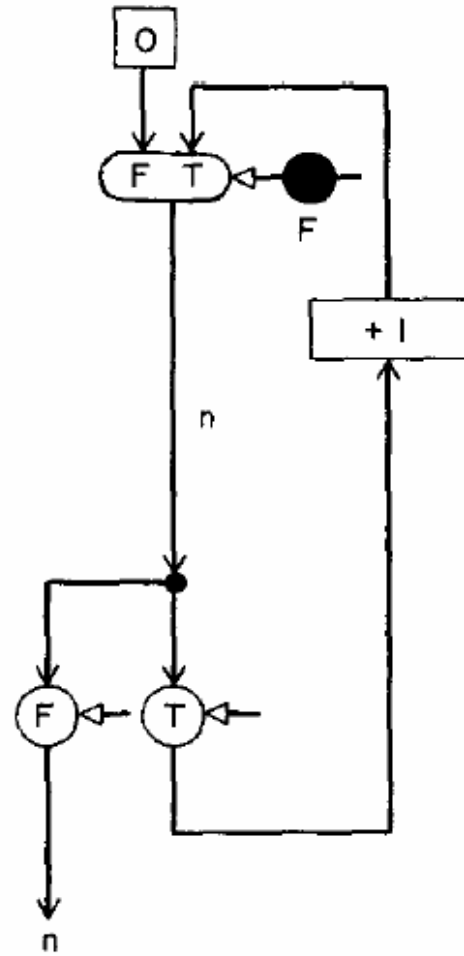
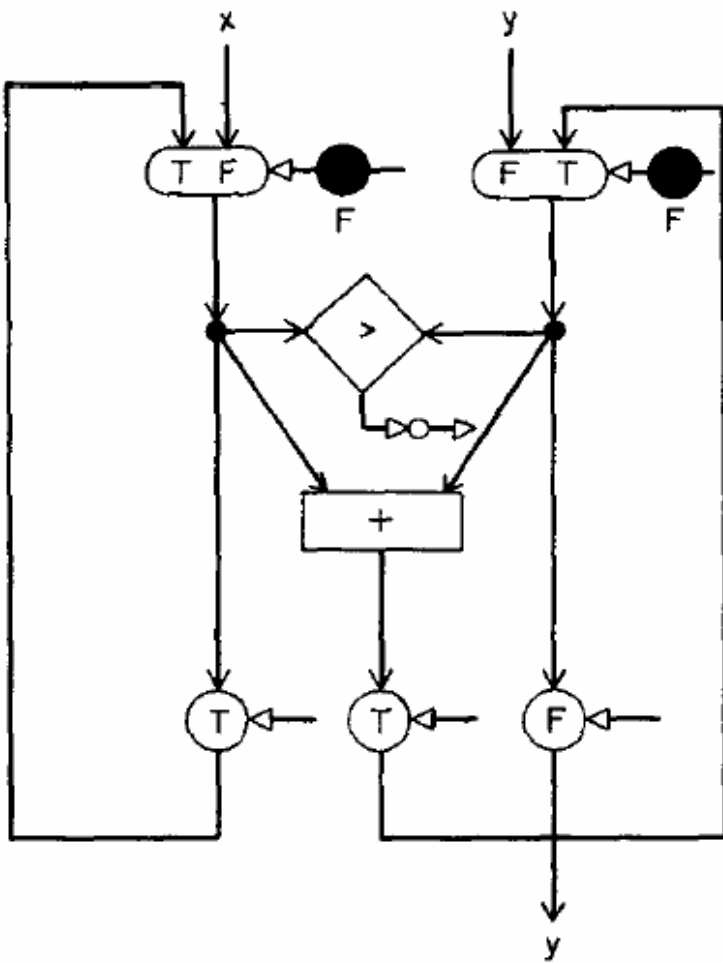


Instruction Cell/Operator

- An Instruction Cell is *enabled* when each of its three registers are full
 - When enabled, the Instruction Cell signals the Arb net



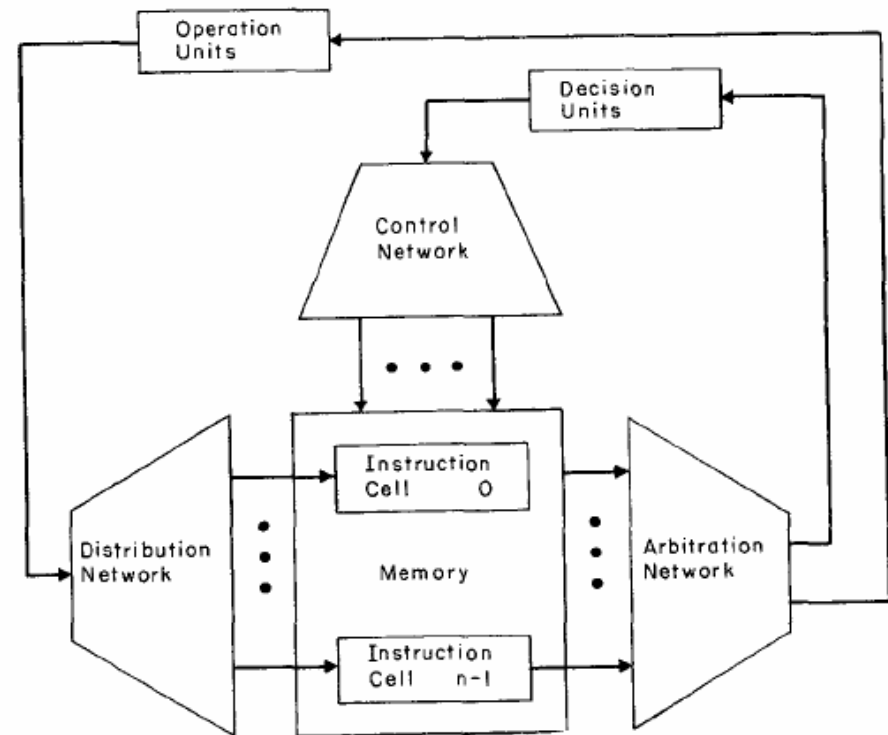
Basic Data Flow Program



```
input y, x  
n := 0  
while y < x do  
  y := y + x  
  n := n + 1  
end  
output y, n
```

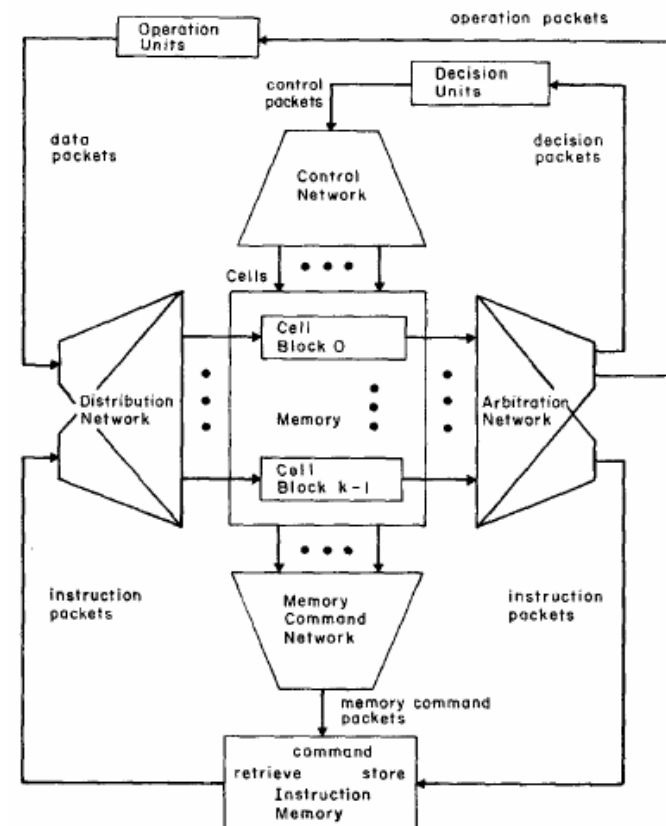
Basic Processor

- Structure is essentially the same
- Packet formats have same character
- Number of instruction cells is a limitation



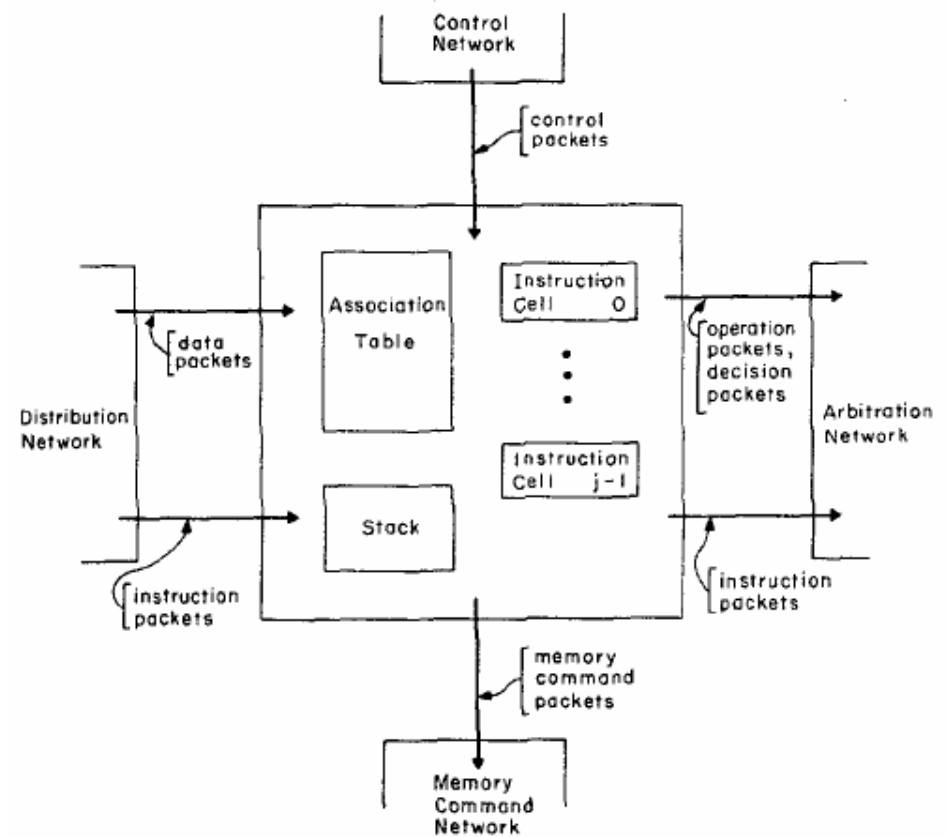
Basic Processor with Memory Hierarchy

- Auxiliary memory allows larger programs
- Cell block must now perform a sort of caching and demand fetching

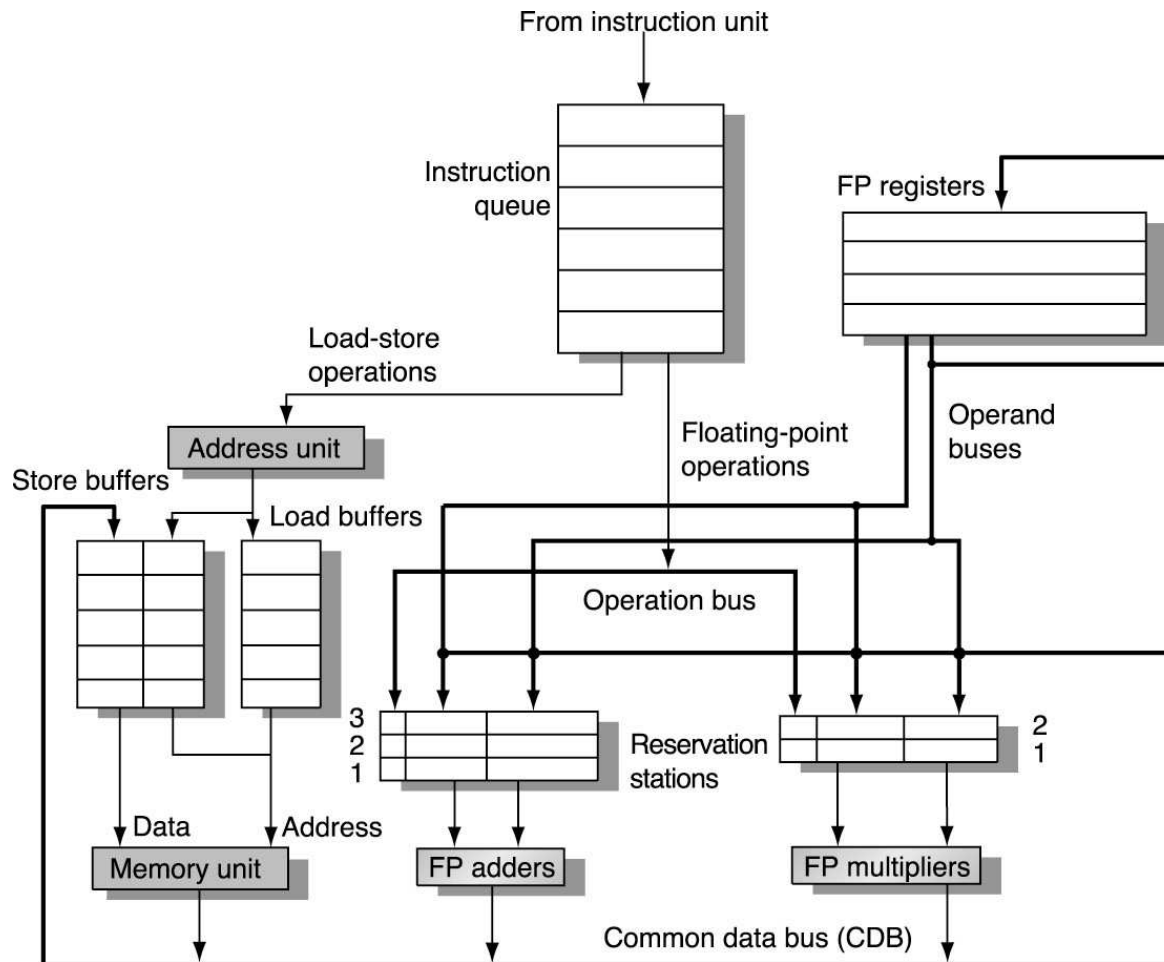


Cell Block Structure

- There are more memory locations than instruction cells
- Assoc table records the status of each cell
- Stack maintains displacement order



Tomasulo's Algorithm



Intellectual Digestion

- What difficulties do you see with using data flow ideas in a system?
- Most data flow examples focus on concurrent computations. Can data flow ideas help tolerate slow memory or fast I/O? If so, how?
- Does the presence of multiple threads increase or decrease the utility of a data flow organization? Why?

Difficulties

- Data flow-friendly languages never achieved critical mass (serial, control flow baggage must be tolerated)
- Serial execution lends itself to step-through debugging; how do you debug a highly concurrent parallel computation?

Data Flow vs. Slow Mem and Fast I/O

- Slow memory
 - Execution can proceed past memory ops whenever independent instructions exist (indirect benefit)
- Fast I/O
 - Perhaps, but no one has done anything yet

Threads and Data Flow

- I think there is merit in considering thread dispatch mechanisms inspired by data flow ideas, but no one has looked at it yet

*Flexible Decoupled Transactional
Memory Support*

Assignment

- Thursday, April 24
 - Project reports and project code due via email
- Thursday, May 1
 - Project presentations due