Multi-Core ßeta Computer

Christopher Celio & Matt Long

6.111 Spring 2007

Project Goals

- Building a functional multi-core computer around the Beta processor (think 6.004 Labs on steroids)
- Visually demonstrate the benefits of multiple cores with "The Game of Life"
- Managing access to memory amongst many Beta processors via a Memory Manager and possibly data caches.

The Game of Life

- A ‘zero-player’ game, user sets initial state, then observes cellular evolution
- For each generation, a cell is either *live* or *dead* based on its number of direct neighbors in the previous generation
- Cells evolve indefinitely on an infinite (in our case 240 x 240) grid

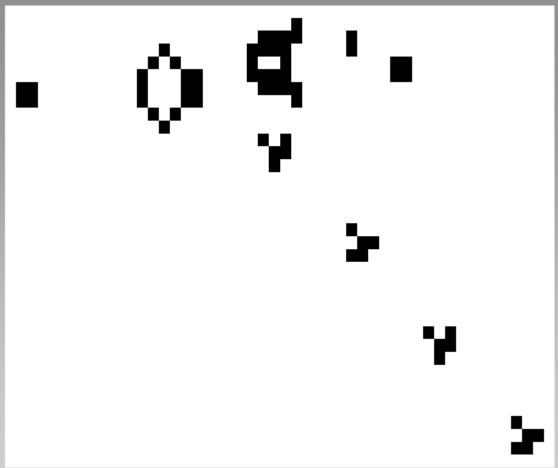
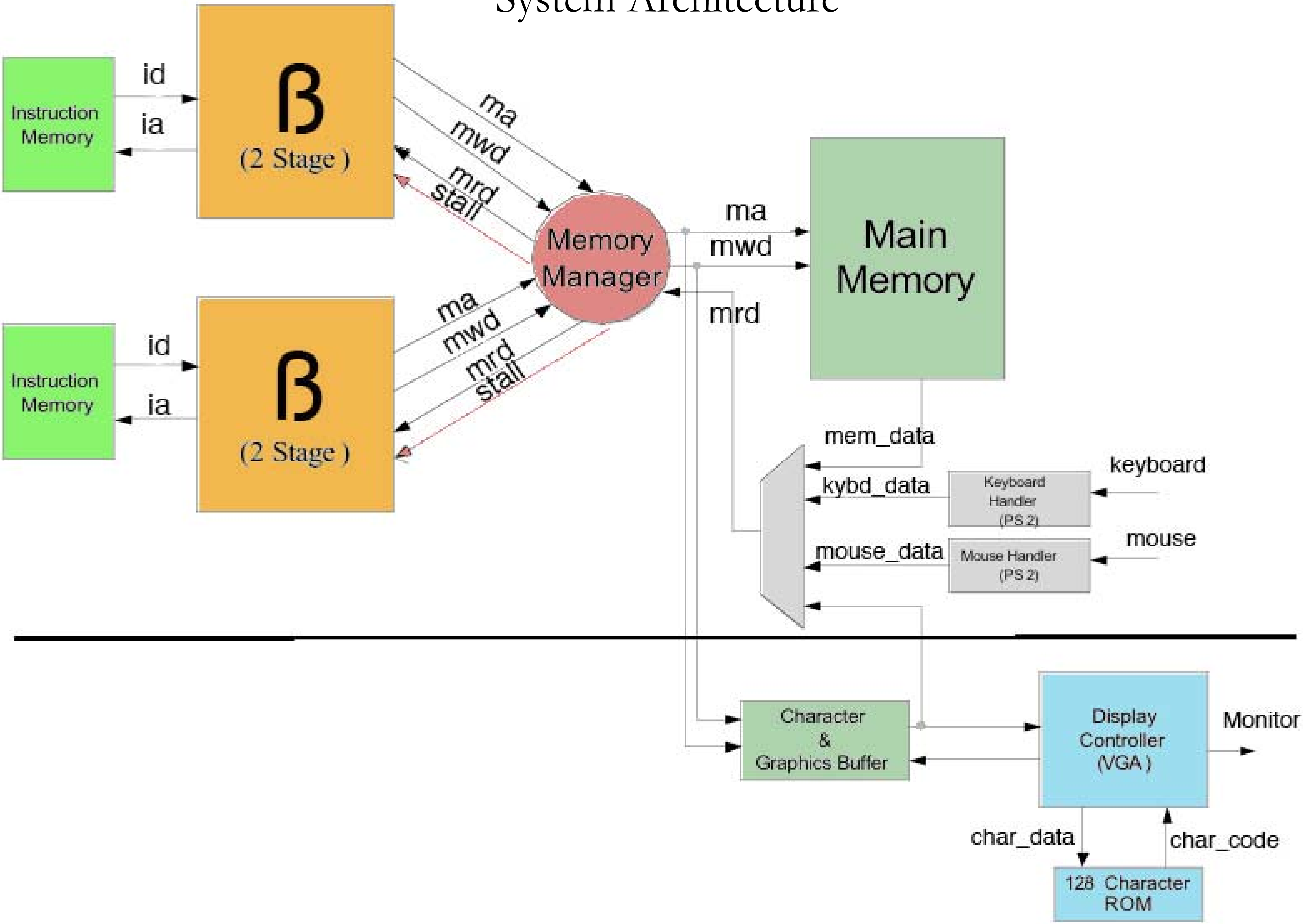
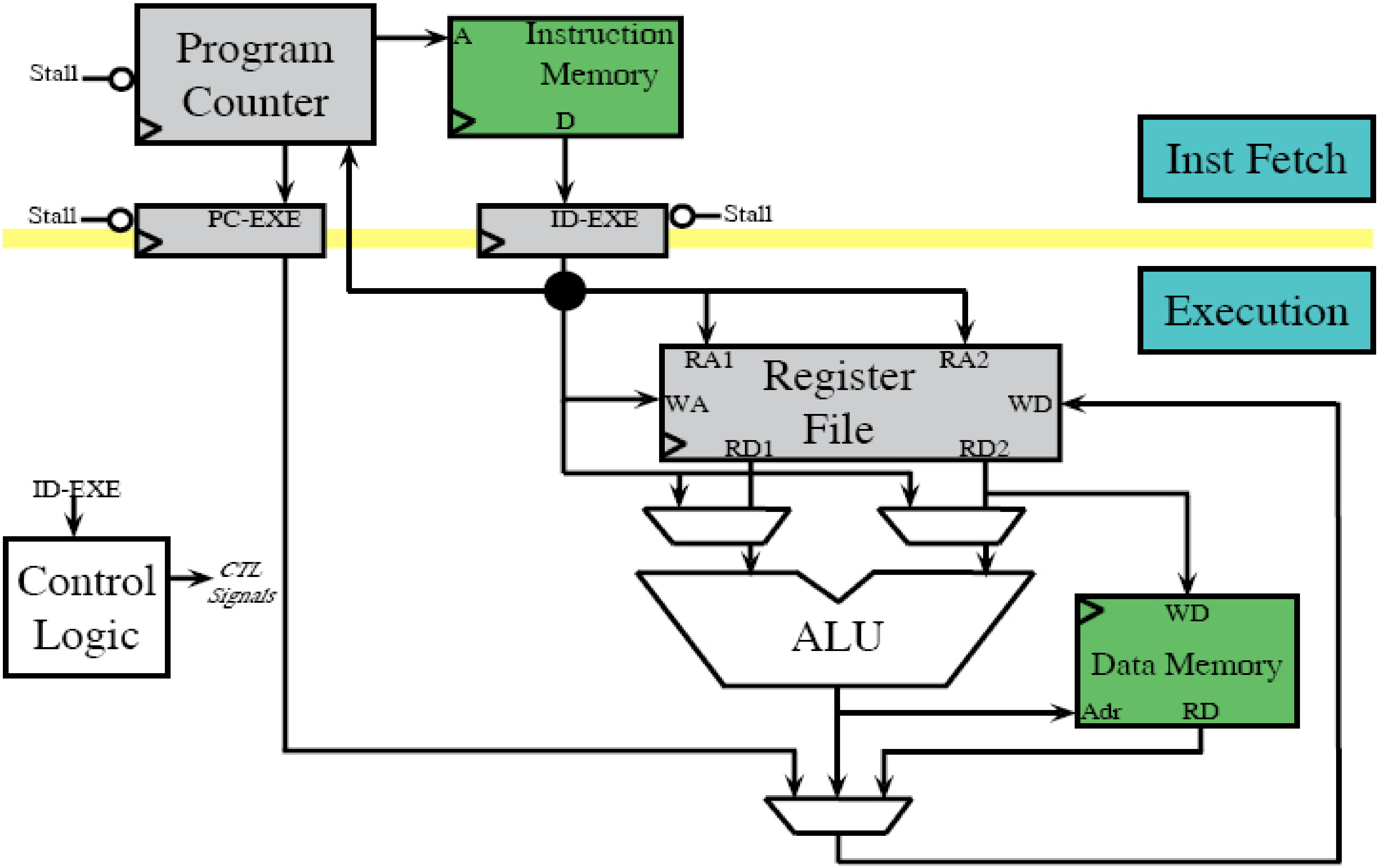


Image courtesy of Wikipedia

System Architecture

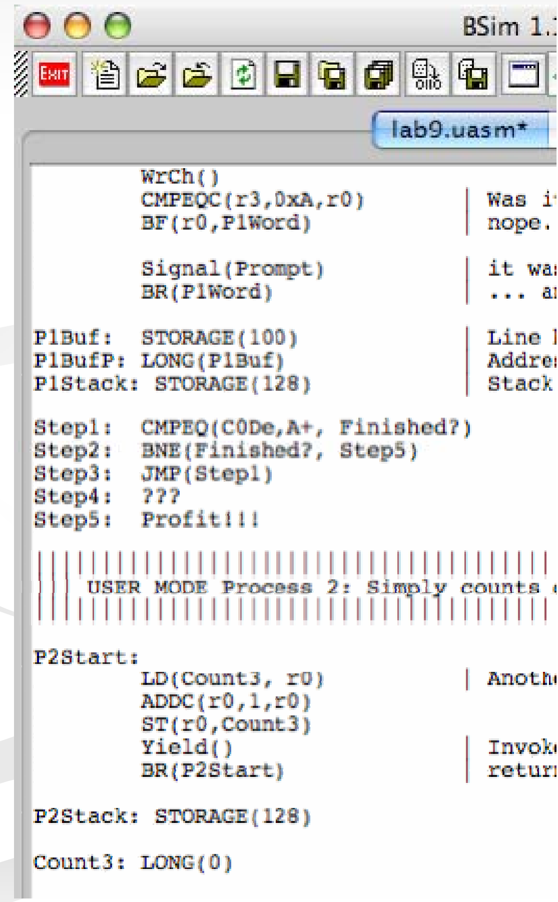


The 2-Stage RISC Harvard Beta Processor



The OS & Software

- Using the 6.004 mini OS as a starting point
- Software written in Assembly and compiled by BSIM
- Python script creates software.v file, which instantiates BRAM and initializes the memory to the Beta machine code



How will the multi-cores communicate?

- Short-answer: software
- Each CPU knows its ID and total_CPU_Count
- Game of Life:
  4 steps per round
  - Compute all cells (all cpus read static image in Memory)
  - Wait (for all to finish)
  - Update all cells (refresh the static image)
  - Wait

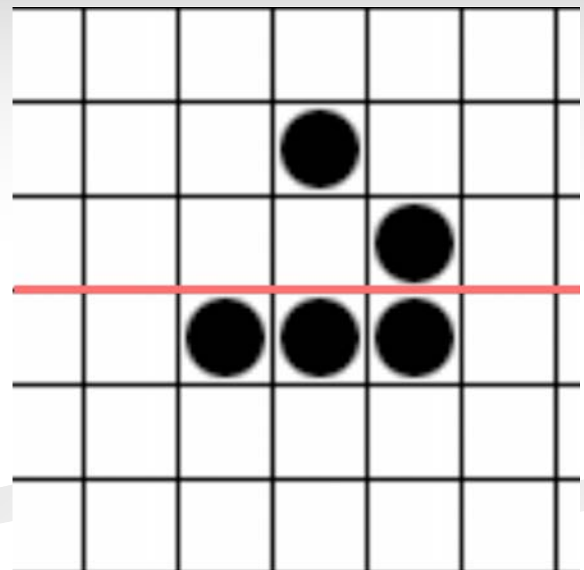
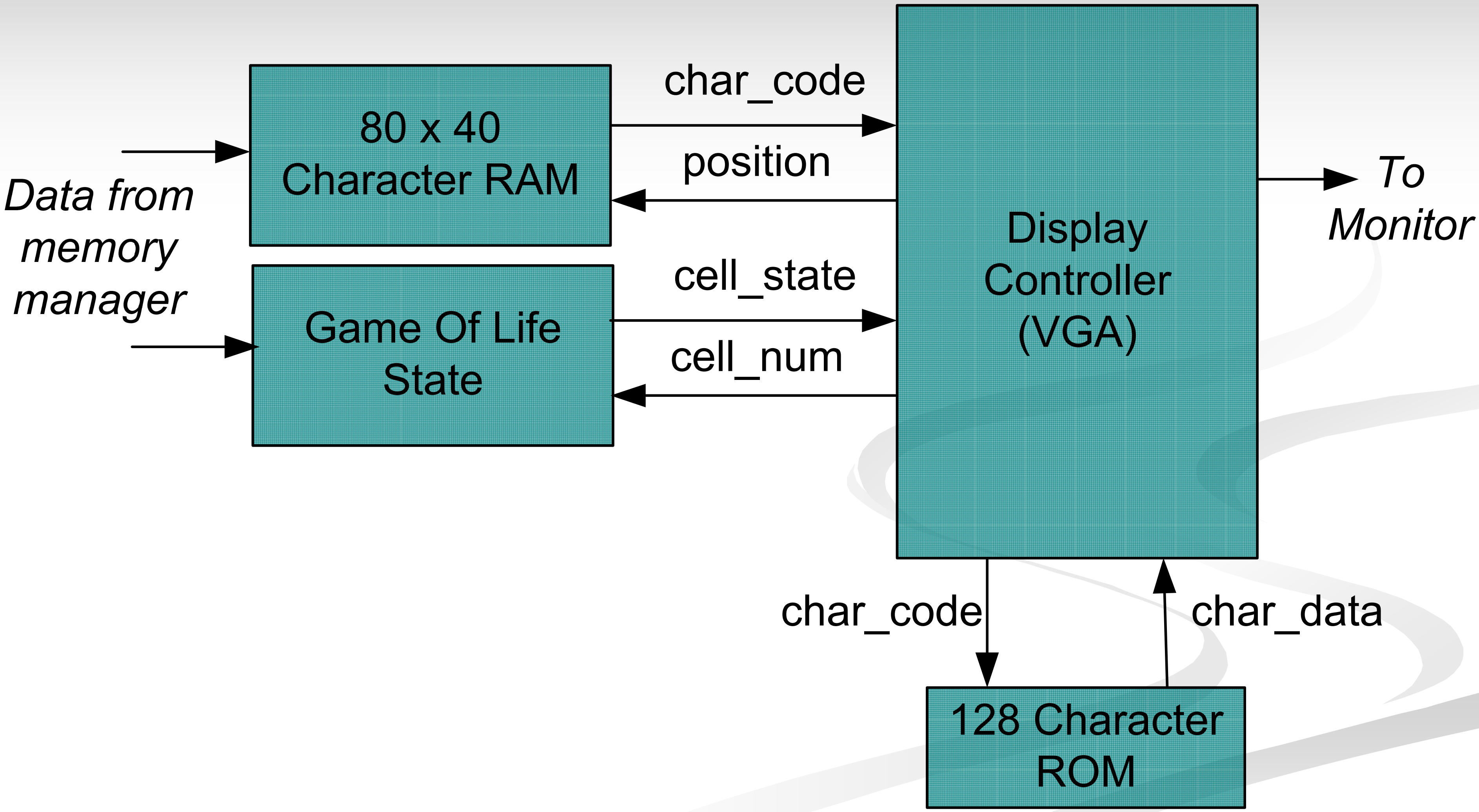


Image courtesy of Wikipedia

Display Controller



Two Display Modes

Console Mode:

- Textual: 80 x 40 character display with prompt
- Interact with the Operating System
- Launch The Game of Life

Game Mode:

- *The Game of Life* cellular grid
- System performance statistics
  - generations/second
  - instructions/second
  - processor usage

VGA Controller Module

- Driven by a clock with twice the frequency of the VGA pixel clock
  - Allows memory access and data processing to occur within each cycle of the pixel clock
- Reads display data from character and/or game state RAMs
- 128 Character ROM

Conclusion

Questions?

Why The Game of Life?

- **Simple, yet computationally intense**: Continually calculate the number of neighbors for each of 50,000+ cells to determine next generation
- **Well suited for a multi-core system**: Time needed to compute each generation decreases linearly as more processors are added
- Fun and interesting way to visually observe the benefits of multiple processors