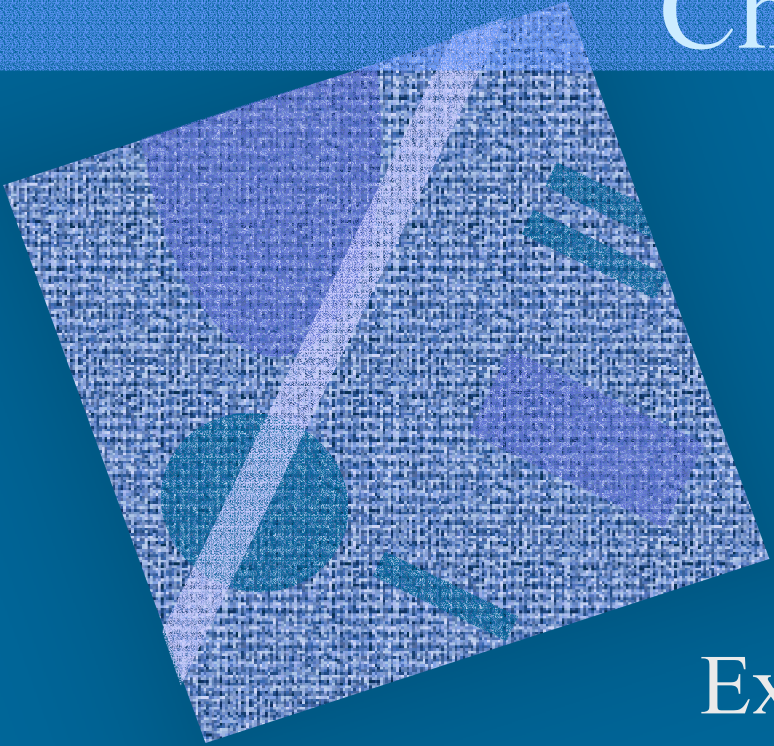
Micro-programmed Control Ch 17



Micro-instructions

Micro-programmed Control Unit

Sequencing

Execution Characteristics

Course Summary

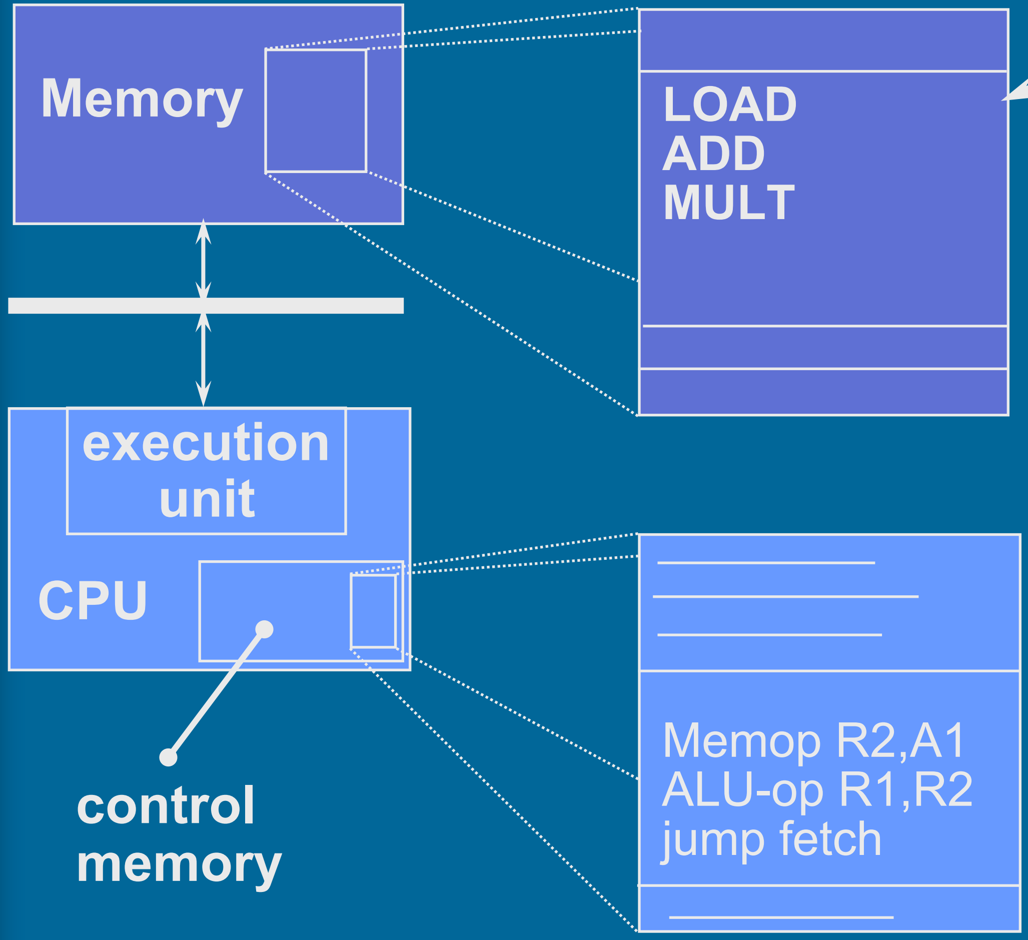
Hardwired Control (4)

- Complex
- Fast
- Difficult to design
- Difficult to modify
  - lots of optimization work done at implementation phase (after design)
  - all optimization work (I.e., most of the work?) must be redone after any changes to design

Micro-programmed Control (3)

- Implement "execution engine" inside CPU
  - execute one micro-instruction at a time
- What to do now?
  - micro-instruction
    - control signals
  - stored in micro-instruction control memory
    - micro-program, firmware
- What to do next?
  - micro-instruction program counter
    - default (?): next micro-instruction
    - jumps or branches?

Machine Instructions vs. Micro-instructions



Machine instructions define a program

Micro-instructions define machine instructions (processor architecture!)

Machine Instructions vs. Micro-instructions (2)

- Machine instruction fetch-execute cycle produces machine instructions to be executed at CPU
- Micro-instruction fetch-execute cycle produces control signals for data path

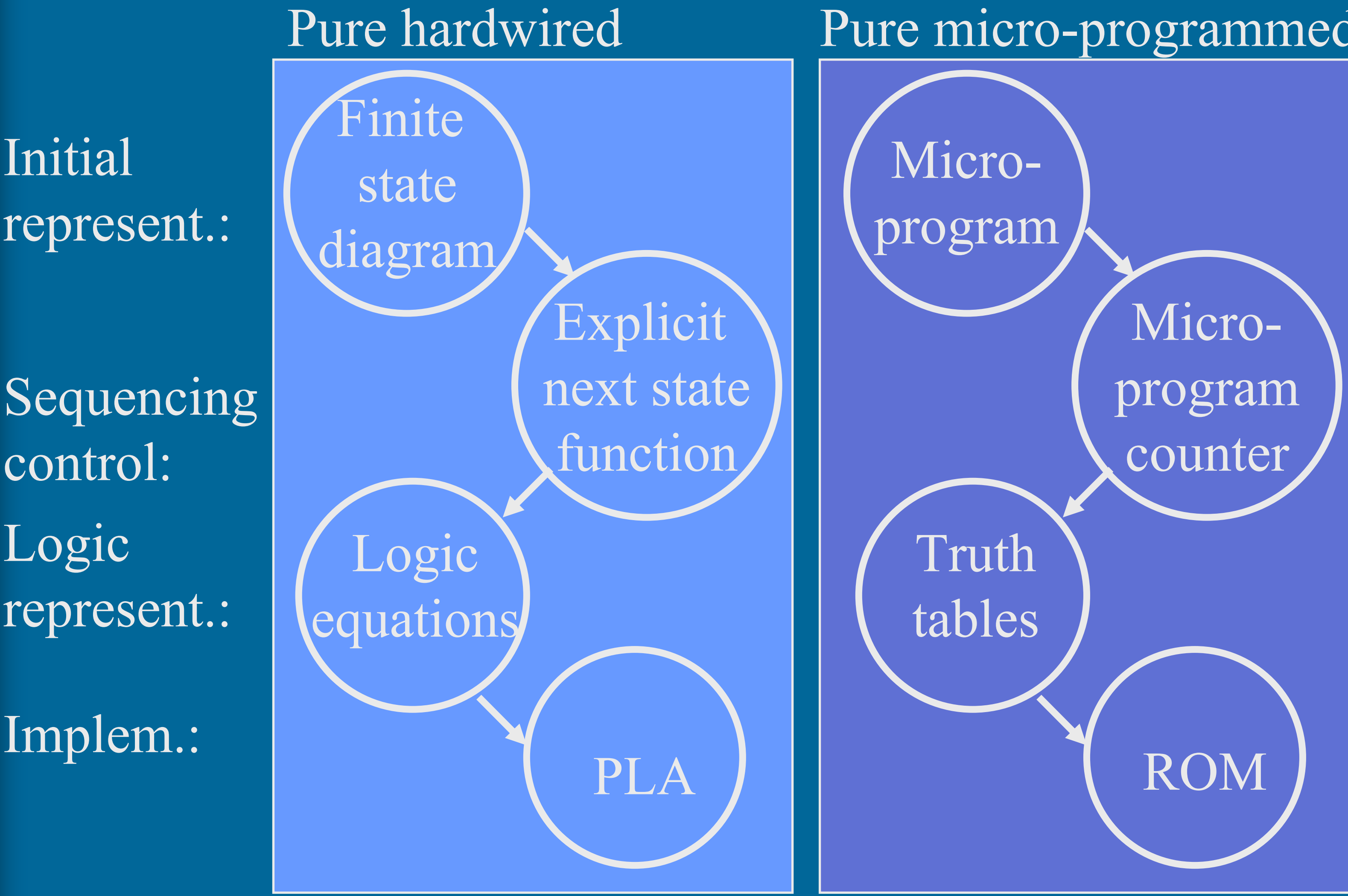
Micro-program (4)

Fig. 17.2

(Fig. 15.2 [Stal99])

- Stored in control memory
- ROM, PROM, EPROM, Flash
- Firmware
- One "subroutine" for each machine instruction
  - one or more micro-instructions
- Defines architecture
  - change instruction set?
    $\Rightarrow$ reload control memory

Hardwired vs. Micro-program Control



Microcode (3)

(Fig. 15.1 (a) [Stal99])

Fig. 17.1 (a)

- Horizontal micro-code
  - control signals directly in micro-code
  - all control signals always there
  - lots of signals $\Rightarrow$ many bits in micro-instruction

(Fig. 15.1 (b) [Stal99])

Fig. 17.1 (b)

- Vertical micro-code
  - each action encoded densely
  - actions need to be decoded to signals at execution time
  - takes less space but may be slower
- Each micro-instruction is <u>also</u> a conditional branch?

Micro-programmed Control Unit (4)

Fig. 17.4

(Fig. 15.4 [Stal99])

- Control Address Register
  - “micro-program PC”
- Control Memory
- Control Buffer Register
  - current micro-instruction
    - control signals
    - next address control
- Sequencing logic
  - select next value for Control Address Reg

Micro-programming (3)

- Simple design
- Flexible
  - –adapt to changes in organization, timing, technology
  - –make changes late in design cycle, or even in the field
- Very powerful instruction sets
  - –use bigger control memory if needed
  - –easy to have complex instruction sets
    - is this good?

Micro-programming (2)

- Generality
  - multiple instruction sets on same machine
  - tailor instruction set to application?
- Compatibility
  - easy to be backward compatible in one family
  - many organizations, same instruction set

Micro-programming (3)

- Costly to implement
  - need tools:
    - micro-program development environment
    - micro-program compiler
- Slow
  - micro-instruction interpreted at execution time
  - interpretation is internal to CPU
  - interpret one instruction at a time
- Interpretation control with hardwired logic?

RISC vs. Micro-programming (8)

- Simple instructions can execute at very high clock rate
- Compilers can produce micro-instructions
  - machine dependent optimization
- Use only simple instructions and addressing mode
- Keep “micro-code” in RAM instead of ROM
- no micro-instruction interpretation logic needed
- Fast access to “micro-code” in RAM via caching
- Skip instruction interpretation of a micro-program and simply compile directly into lowest language of machine?
- $\Rightarrow$ Compile to “micro-code” and use hardwired control for RISC (e.g., Pentium II)

Micro-program Sequencing (3)

- Two address format
  - most often use the subsequent address
    - waste of space to store it most of the time?
  - conditional branch address

Fig. 17.6 (Fig. 15.6 [Stal99])

- One address format
  - (Conditional) branch address

(Fig. 15.7 [Stal99]) Fig. 17.7

- Variable format
  - only branch micro-instructions have addresses
    - branch instruction do not contain control signals
  - waste of time many times?

Micro-instruction Explicit Address Generation

- Addresses explicitly present
  - Two-field
    - select one of them
  - Unconditional branch
    - jump to this one
  - Conditional branch
    - select this one or default

Micro-instruction Implicit Address Generation

- Addresses not explicitly present
  - Mapping
    - map opcode in machine instruction into micro-instruction address

(Fig. 15.9 [Stal99])

Fig. 17.9

  - Addition
    - higher order bits directly from opcode
    - lower order bits based on current status and tag bits, or fields in current microinstruction
  - Residual Control
    - return from micro-program subroutine

Micro-instruction Encoding

- Usually a compromise between pure horizontal and vertical formats

Fig. 17.11 (Fig. 15.11 [Stal99])

  - optimize on space with encoding multiple signals into a set of fields
    - each field defines control signals for certain separat actions
    - mutually exclusive actions are encoded into the same field
  - make design simpler by not using maximum encoding

Micro-instruction Encoding (2)

- Functional encoding
  - each field controls some function
    - load accumulator
    - load ALU operands
    - compute next PC
- Resource encoding
  - each field controls some resource
    - ALU
    - memory

Different Micro-instruction Sets for a specific ”Simple Machine” (3

(Fig. 15.12 [Stal99])

Fig. 17.12

- Micro-instruction types
  - 3 register transfers, 2 mem ops, 5 ALU ops, 3 seq. ops
- Vertical format

type	operation	reg

  - 3 bits for type, 3 bits for operation
  - 2 bits for reg select (max 4 regs)

(Fig. 15.12(a) [Stal99])

Fig. 17.12 (a)

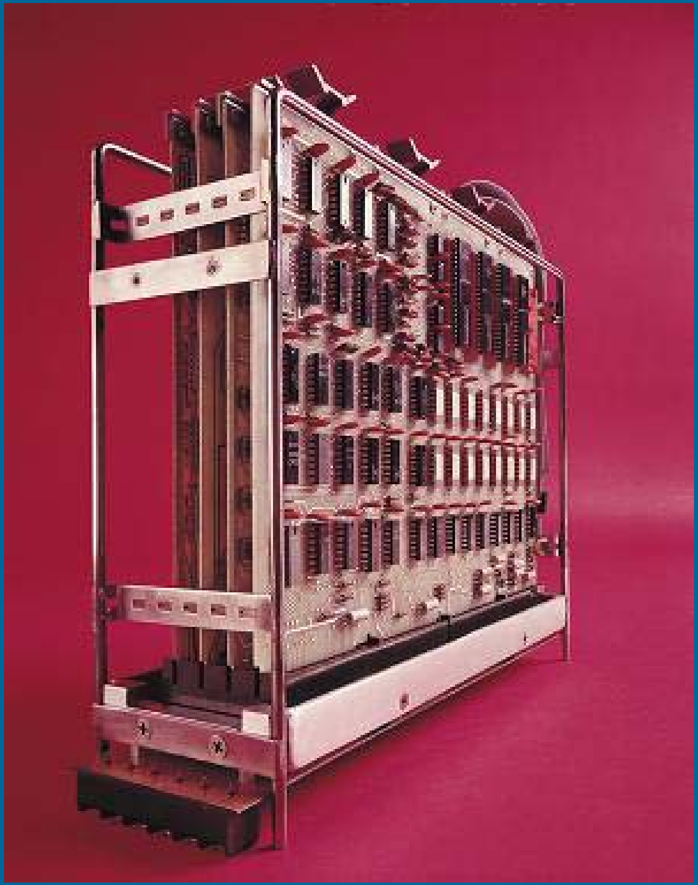
- Horizontal format

  - 2 bits for reg transfers (3 ops + “none”)
  - 2 bits for mem ops (2 ops + “none”)
  - 2 bits for seq. ops (3 ops + “none”)
  - 3 bits for ALU ops (5 ops + “none”)
  - 2 bits for reg select + 8 bits for constant

Fig. 17.12 (b)

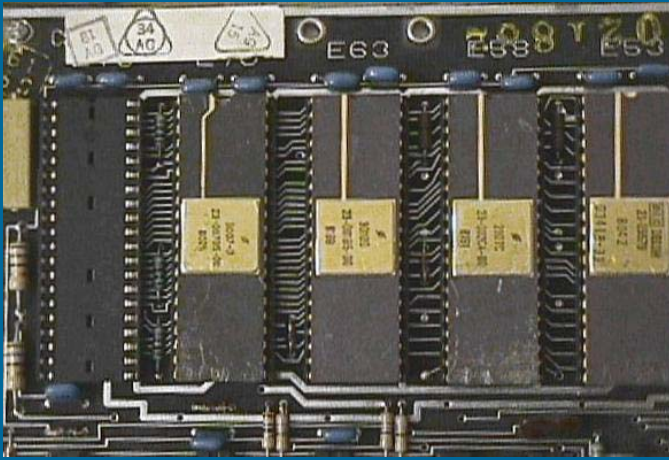
(Fig. 15.12(b) [Stal99])

LSI-11 Single Board Processor



LSI-11 (PDP-11) (5)

- Three-chip single board processor
  - data chip
    - 26 8-bit regs
      - 8 16-bit general purpose regs,
      - PWS, MAR, MBR, ...
    - 8-bit ALU
      - (at least) 2 passes needed for 16-bit reg ops
  - control chip
  - control store chip (2 of them?)
    - 22 bit wide control mem for micro-instructions
  - connected by micro-instruction bus



(Fig. 15.14 [Stal99])

Fig. 17.14

Fig. 17.13

(Fig. 15.13 [Stal99])

LSI-11 Micro-instruction Set (2)

- Implements PDP-11 instruction set architecture for LSI-11 hardware
  - e.g., PDP-11 16 bit ALU vs. LSI-11 8-bit ALU
- 22 bit wide, extremely vertical set
  - 4 bits for special functions
  - 1 bit for testing interrupts
  - 1 bit for "micro-subroutine return"
  - 16 bits for variable format micro-ops
    - jump, cond. branch, literal ops, reg ops
    - ALU, logical, general, I/O ops

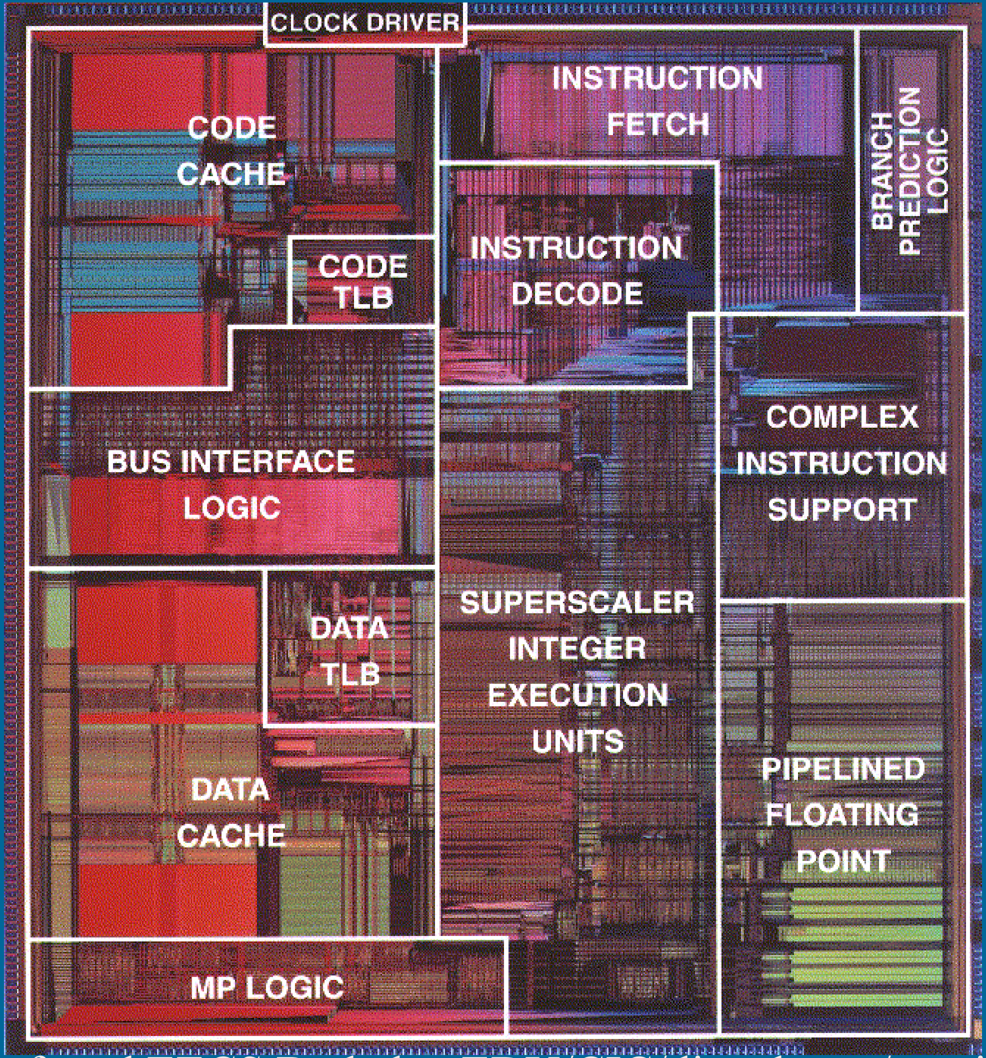
Fig. 17.15

(Fig. 15.15 [Stal99])

Table 17.5

(Tbl 15.5 [Stal99])

-- End of Chapter 17 --
-- Micro-programmed Control --



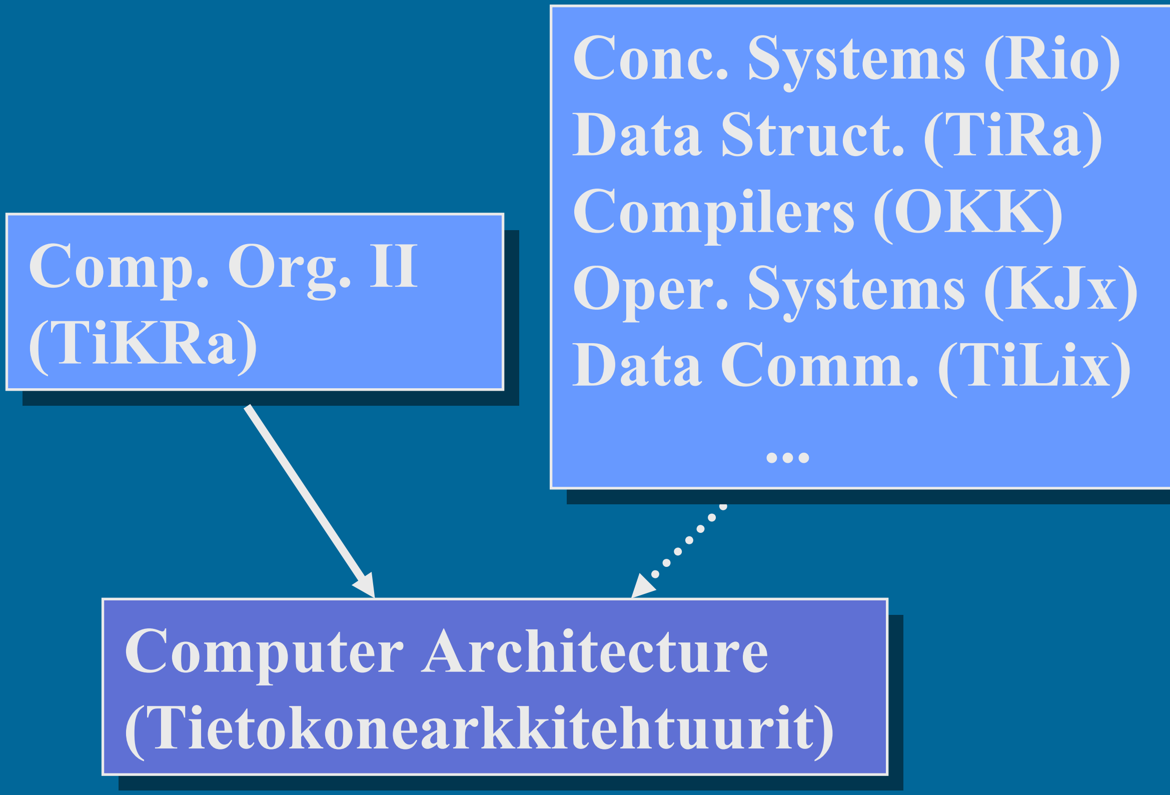
http://infopad.EECS.Berkeley.EDU/CIC/die_photos/pentium.gif

Summary (11)

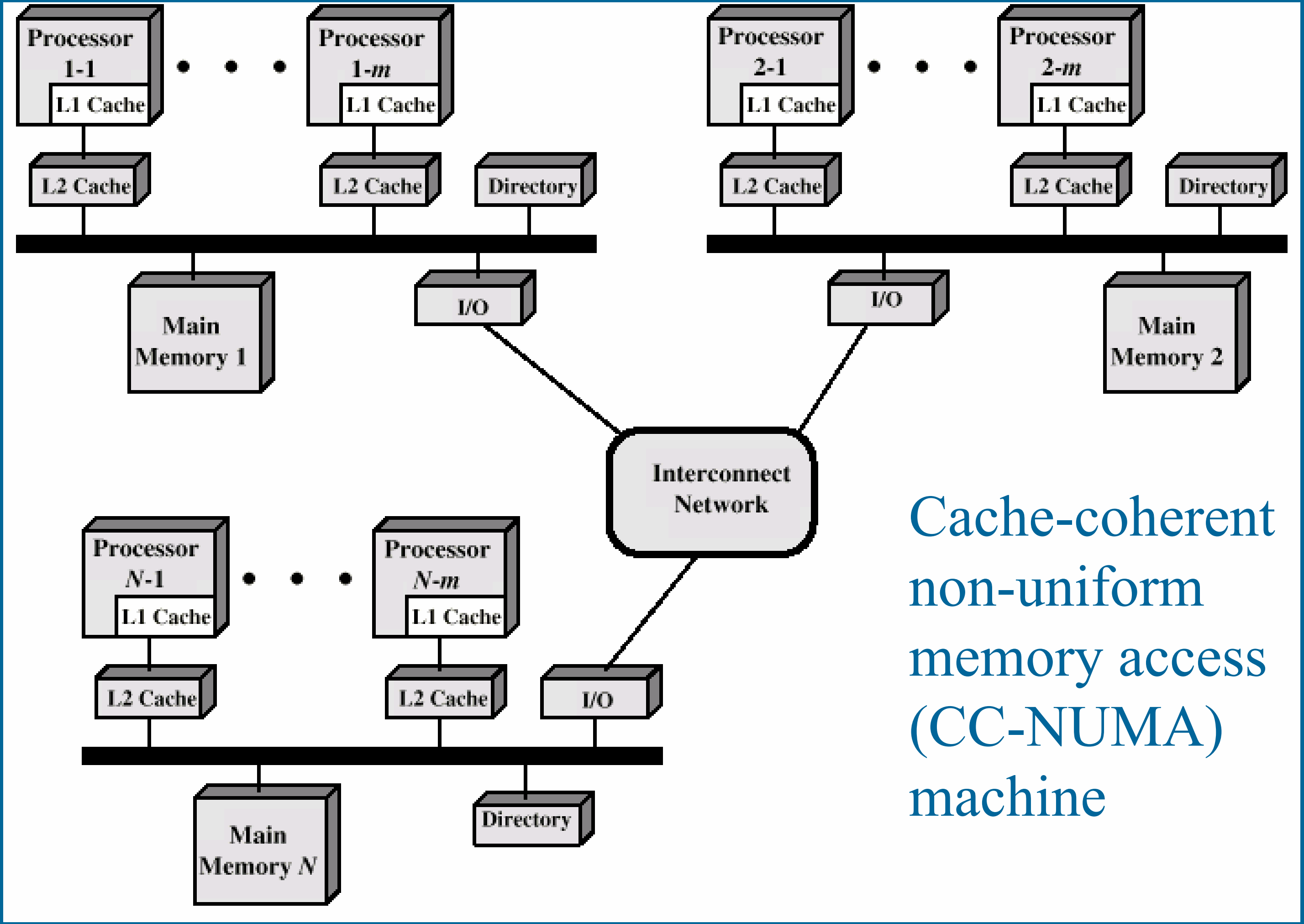
- How clock signals cause instruction executions?
- Low level stuff
  - gates, basic circuits, registers, memory
- Cache
- Virtual memory & TLB
- ALU, Int & FP arithmetic's
- Instruction sets
- CPU structure & pipelining
- Branch prediction, limitations, hazards, issue
- RISC & superscalar processor, name dependencie
- IA-64 & Crusoe
- Hardwired & micro-controlled control

Want to Know More?

- Read the text book completely
- 58070-8 Computer Architecture (4 cr)



-- The End of Comp Org II --



Cache-coherent non-uniform memory access (CC-NUMA) machine

(Fig. 18.11)